Module M2102

ADMINISTRATION SYSTÈME

Gestion de réseaux d'utilisateurs
Introduction aux systèmes virtualisés

Camille Coti [1]
camille.coti@iutv.univ-paris13.fr
IUT de Villetaneuse - Département R&T - 2014-2015

1. Quelques petites corrections faites en 2019 par Luca Saiu (http://ageinghacker.net).

Table des matières

1 Introduction

Ce polycopié est le support de cours du module M2102 du DUT Réseaux et Télécommunications. Ce module porte sur l'administration de systèmes d'exploitation. Il est composé de deux sous-parties : l'administration d'un système multi-utilisateur, comprenant la gestion des utilisateurs et les limites que l'on peut leur fixer, la sûreté du stockage et la métrologie, et les notions de virtualisation, ses concepts de base reposant sur la notion de noyau de système d'exploitation, et une introduction aux fondamentaux du Cloud Computing.

Ce module est composé de trois cours magistraux et de six séances de TP. Il est attendu que vous lisiez des parties du polycopié pour préparer les TP, et que vous le gardiez avec vous lors des TP pour vous y référer.

Le polycopié ne constitue pas un manuel exhaustif d'administration système, mais un support de cours pour ce module. Il est là pour vous présenter des notions et des outils, que vous approfondirez le temps venu lorsque vous aurez à en utiliser l'un ou l'autre. La documentation en ligne (les pages du `man`) et les commentaires des fichiers de configuration indiqués sont un très bon départ pour se documenter. En TP, vous aurez notamment à les lire et à les comprendre.

Première partie
Administration système

2 Gestion des utilisateurs sur un système multi-utilisateurs

La plupart des systèmes actuels sont dits *multi-utilisateurs* : ils peuvent être utilisés par plusieurs utilisateurs, et chaque utilisateur doit avoir son propre compte. Pour des raisons basiques de sécurité (au-delà de la confidentialité), il est déconseillé que plusieurs utilisateurs partagent le même compte.

Pour permettre cela, les systèmes d'exploitation multi-utilisateurs disposent de fonctionnalités de gestion des utilisateurs et de cloisonnement, au moins basique et minimal, de ce que chaque utilisateur peut faire sur le système.

On peut obtenir des informations sur un utilisateur avec la commande `id`, qui affiche l'iud et les groupes auxquels un utilisateur appartient, ou avec la commande `finger`, qui affiche plus d'informations sur l'utilisateur.

2.1 Utilisateurs locaux

Les utilisateurs locaux sont créés sur le système local et n'ont d'existence que sur celui-ci. Les informations qui leur sont relatives sont stockées sur le système et ils ne peuvent se connecter et effectuer des opérations que sur celui-ci.

2.1.1 Fichiers relatifs aux comptes des utilisateurs locaux

Les informations des comptes des utilisateurs locaux sont stockées dans deux fichiers : `/etc/passwd` et `/etc/shadow`. Ces deux fichiers appartiennent au super-utilisateur ; `/etc/passwd` est lisible par tout le monde, tandis que `/etc/shadow` n'est lisible par le super-utilisateur et les membres du groupe shadow. Ils ne sont évidemment modifiables que par le super-utilisateur. Contrairement à ce que leur nom évoque, c'est le fichier `/etc/shadow` qui contient les mots de passe et non pas le fichier `/etc/passwd`.

```
coti@maximum:~$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   1762 févr. 14 14:51 /etc/passwd
-rw-r----- 1 root shadow 1196 févr. 14 14:51 /etc/shadow
```

Le fichier `/etc/passwd` contient les informations relatives aux comptes eux-mêmes. Il contient une ligne par compte utilisateur local, chaque ligne étant composée de sept champs séparés par le symbole " :" :

- `username` : le login (nom de connexion) ;
- `passwd` : le mot de passe, non-affichable. C'est en réalité un héritage historique : il y avait autrefois le mot de passe crypté à cet endroit. Ce champ contient désormais un `x`, qui ne doit surtout pas être enlevé ;
- `IUD` : le numéro d'utilisateur (user id, ou *uid*) ;
- `GID` : le numéro du groupe primaire (group id, ou *gid*) de l'utilisateur ;
- `full_name` : le nom complet de l'utilisateur et d'éventuelles remarques ;
- `directory` : le chemin vers le répertoire personnel ;
- `shell` : le shell de connexion, c'est-à-dire l'interpréteur de commandes qui va être exécuté pour recevoir les commandes de l'utilisateur.

Ce fichier peut être modifié à la main ou en utilisant des outils prévus pour effectuer les changements demandés. Par exemple, `chsh` permet de changer le shell de connexion d'un compte, `chfn` permet de modifier le nom complet, etc.

Ce fichier ne contient pas vraiment d'information sensible (il ne contient pas de mot de passe) et doit être lisible par tous les utilisateurs. C'est ce qui est utilisé, par exemple, pour faire la traduction

entre les uids et les noms d'utilisateurs. Lorsque l'ont fait `ls -l` sur un fichier, on voit le nom de son propriétaire. Si le fichier `/etc/passwd` n'est pas lisible, on verra l'uid de son propriétaire. Cependant, il ne doit pas être modifiable par tout le monde.

Par exemple, voici un extrait d'un fichier `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
figatellix:x:1000:1001:Figatellix,,,:/home/figatellix:/bin/bash
```

La première ligne concerne l'utilisateur `root`, ou le super-utilisateur. On voit que son iud et son gid sont tous les deux à 0 (valeur particulière pour cet utilisateur). Son nom complet est "root". Son répertoire personnel est `/root`, et son shell de connexion est `/bin/bash`.

L'utilisateur `www-data` a lui pour répertoire personnel le répertoire `/var/www`. C'est l'utilisateur sous l'identité duquel tournera la serveur HTTP. Son shell de connexion est `/bin/sh`.

Le fichier `/etc/shadow` contient des informations de mot de passe des comptes. C'est ici que sont stockés les mots de passe, cryptés, bien entendu, avec le programme `crypt`. Il contient une ligne par compte utilisateur local, chaque ligne étant composée de neuf champs séparés par le symbole " :" :

- `username` : le login (nom de connexion) ;
- `passwd` : le mot de passe, crypté ;
- `last` : le nombre de jours entre le 1er janvier 1970 et la date de dernier changement du mot de passe ;
- `may` : le nombre de jours avant que le mot de passe puisse être changé (plus utilisé actuellement : on met 0 dans ce champ) ;
- `must` : le nombre de jours avant que le mot de passe doive être changé (rarement utilisé) ;
- `warn` : le nombre de jours durant lesquels l'utilisateur est prévenu de l'expiration de son mot de passe ;
- `expire` : le nombre de jours entre l'expiration du mot de passe et la fermeture du compte ;
- `disable` : la date de fermeture du compte, en nombre de jours depuis le 1er janvier 1970 ;
- `reserved` : champ réservé pour un éventuel usage ultérieur.

Ce fichier contenant les mots de passe des utilisateurs, même sous forme cryptée, il ne doit pas être lisible par tous les utilisateurs. Cependant, certains programmes (comme `xclock`) peuvent avoir besoin de lire son contenu. Les utilisateurs exécutant ces programmes doivent être membres d'un groupe particulier, le groupe `shadow`, qui a la possibilité de lire ce fichier.

Par exemple, voici un extrait d'un fichier `/etc/shadow` (les mots de passe cryptés ont été modifiés et raccourcis) :

```
root:$6$6sjgOpPD$FxuhwgYeczgWuAIU7qjuw81:16104:0:99999:7:::
www-data:*:16104:0:99999:7:::
backup:*:16104:0:99999:7:::
list:*:16104:0:99999:7:::
irc:*:16104:0:99999:7:::
gnats:*:16104:0:99999:7:::
figatellix:$6$ZzwWAP5z$DzF2Qf8pib3CY88EqeBlQz6KNC1:16104:0:99999:7:::
```

On constate que certains utilisateurs ont un astérisque ("*") à la place du mot de passe. Ce champ peut également être remplacé par un point d'exclamation (" !"). Il signifie qu'il est impossible de se connecter à ces comptes en utilisant un mot de passe.

2.1.2 Types d'utilisateurs

Il existe plusieurs types d'utilisateurs locaux. Les possibilités offertes à ces comptes ne sont pas les mêmes, et chacun doit être utilisé dans la situation pour laquelle il a été prévu. L'accès à une machine étant une question particulièrement critique dans la sécurisation d'un système, chaque utilisateur ne doit pouvoir faire que ce qu'il a besoin de faire sur ce système. Par conséquent, pour chaque type d'usage de ce système il existe une catégorie d'utilisateur.

Le super-utilisateur (root) : il a (presque) tous les droits sur le système. Par conséquent, il faut l'utiliser avec parcimonie et uniquement lorsque c'est absolument nécessaire. Seuls des systèmes d'exploitation orientés vers la sécurité (par exemple SELinux) limitent ce que le super-utilisateur a le droit de faire sur le système. Son user id est particulier : il a toujours l'iud 0.

Il peut notamment créer et supprimer des utilisateurs, changer les droits associés à *tous* les fichiers... Il est donc particulièrement vulnérable aux attaques et doit être protégé le mieux possible. Sur certains systèmes, il n'est par exemple pas possible de se connecter à distance en tant que root. De manière générale, il faut le réserver aux opérations d'administration de la machine.

Les comptes utilisateurs : ce sont, comme leur nom l'indique, les comptes des utilisateurs du système.

Les comptes système : ce sont des comptes particuliers qui n'appartiennent pas à une personne physique, mais qui sont utilisés par le système pour des tâches bien précises, comme faire tourner un service.

En effet, certains services ont besoin de permissions particulière (accès à certains répertoires, etc). Par conséquent, deux possibilités existent : les faire tourner sous l'identité du super-utilisateur, ou sous une identité spécifique utilisée uniquement pour ce service. La première possibilité est risquée, car si le service est corrompu, le programme malveillant dispose alors de l'identité du super-utilisateur. Pour éviter cela, on crée des utilisateurs spécifique pour ces services : un utilisateur pour le serveur d'impression, un utilisateur pour le serveur HTTP, etc.

Ces comptes existent uniquement pour faire tourner ces programmes. Par conséquent, ils n'ont aucune raison de devoir se connecter au système. Les comptes systèmes doivent donc être configurés de telle sorte qu'il soit impossible de se connecter sur le système en utilisant leur identité. Ils n'ont donc pas de répertoire personnel ni de shell de login.

Extrait du fichier `/etc/passwd` : on peut voir la différence entre ces types de comptes dans le fichier `/etc/passwd` (voir section 2.1.1 pour le détail de ce fichier). Par exemple, voici un extrait d'un fichier `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
avahi:x:106:115:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
figatellix:x:1000:1001:Figatellix,,,:/home/figatellix:/bin/bash
```

La première ligne donne les informations concernant le compte du super-utilisateur ou `root`. On constate que son uid est bien 0, ainsi que son gid.

La deuxième ligne concerne l'utilisateur `avahi`. Il s'agit d'un compte système. On voit que son shell de connexion est `/bin/false`, qui n'est en réalité pas un interpréteur de commande mais un programme renvoyant simplement la valeur false (ou 1). Il est donc impossible de se connecter au système sous cette identité car aucun interpréteur de commande ne lui est lié. On a également vu dans la section 2.1.1 qu'il était possible de rendre impossible la connexion à un compte avec un mot de passe en remplaçant le champ dédié au mot de passe par un astérisque ou un point d'exclamation dans le fichier `/etc/shadow`.

La troisième ligne concerne l'utilisateur `figatelix`. il a bien un interpréteur de commande (`/bin/bash`) et un répertoire personnel (`/home/figatellix`). Il s'agit d'un compte utilisateur. Son iud est 1000 et son gid est 1001.

2.2 Comptes globaux sur le réseau

Les comptes utilisateurs locaux, vus à la section 2.1, ont une portée limitée au système local. Pour permettre à des utilisateurs de se connecter sur un ensemble de machines du réseau, on utilise des systèmes de comptes *réseaux*. Un utilisateur réseau sera connu de toutes les machines utilisant ce système.

On peut citer les systèmes d'annuaire comme NIS, LDAP, Kerberos ou Active Directory sous Windows.

Ces systèmes sont des services d'*annuaires* : ils maintiennent des informations sur les comptes et fournissent ces informations aux clients qui le demandent. Par conséquent, les informations sur les comptes sont centralisées au niveau du serveur. Les machines clientes ne disposent pas localement de ces informations.

Lorsqu'un utilisateur se connecte à une machine, le schéma de fonctionnement de base est le suivant :

— L'utilisateur saisit son login et son mot de passe ;
— La machine locale envoie le tout (bien sûr sous forme cryptée) au serveur d'annuaire ;
— Le serveur d'annuaire vérifie si l'association entre le login et le mot de passe est correcte et répond à la machine ;
— La machine accepte ou refuse la connexion de cet utilisateur.

Il est possible de définir des règles plus précises, par exemple de n'autoriser que certains utilisateurs sur certaines machines.

NIS (Network Information Service) : NIS est utilisé uniquement entre machines Unix, à l'échelle d'un réseau local. Pour gérer les utilisateurs réseaux, le serveur NIS utilise ses fichiers `/etc/passwd` et `/etc/shadow` locaux. À partir de ces fichiers, il génère des *maps* utilisées par le serveur NIS et qui contiennent les informations des utilisateurs.

LDAP (Lightweight Directory Access Protocol) : LDAP est à l'origine un protocole d'accès à des données et de modifications de celles-ci. Il propose un grand nombre de fonctionnalités et est, par conséquent, plus complexe (mais plus complet !) que NIS. Il est utilisable par des clients sous différents systèmes d'exploitation, et n'est donc pas réservé à Unix. Les données sont organisées dans le répertoire LDAP selon une structure arborescente (le Directory Information Tree). Le seul accès aux données de LDAP est soumis à un contrôle d'accès : n'importe qui ne peut pas accéder à ces données.

Kerberos est un protocole d'authentification basé sur un échange de clés. Un utilisateur s'authentifie auprès d'un serveur d'authentification en lui fournissant son login et son mot de passe. Celui-ci lui renvoie un message chiffré par une clé secrète, qui doit être déchiffré en utilisant le mot de passe fourni par l'utilisateur. Si le mot de passe est correct, il peut générer un ticket Kerberos qui sera utilisé par l'utilisateur (via les clients qu'il utilise) pour s'identifier sur le réseau.

2.3 Groupes d'utilisateurs

Les utilisateurs sont rassemblés dans des *groupes*. On peut alors définir des permissions à l'échelle d'un groupe, afin de les donner à un ensemble d'utilisateurs. Par exemple, on peut donner accès à un fichier en lecture à groupe particulier et ne pas l'autoriser pour le reste des utilisateurs.

2.3.1 Groupe primaire ou groupe secondaire

Un utilisateur fait obligatoirement partie d'un groupe. Il peut par ailleurs faire partie de plusieurs autres groupes.

Groupe primaire : un utilisateur est membre d'au moins un groupe : c'est son groupe primaire. C'est l'identifiant de ce groupe que l'on trouve dans le fichier `/etc/passwd`. Lorsqu'un fichier est créé, il appartient à cet utilisateur et à son groupe primaire.

Groupes supplémentaires : un utilisateur peut appartenir à d'autres groupes : ce sont ses groupes secondaires.

La commande `id` permet de connaître les groupes auxquels un utilisateur appartient :

```
coti@maximum:~$ id figatellix
uid=1000(figatellix) gid=1001(figatellix) groupes=24(cdrom),25(floppy),29(audio),
30(dip),44(video),46(plugdev),104(scanner),109(bluetooth),111(netdev),1001(figatellix)
```

On voit que l'utilisateur `figatellix` a pour groupe primaire le groupe `figatellix` de gid 1001, et appartient également aux groupes `cdrom`, `floppy`, `audio`, `dip`, `video`, `plugdev`, `scanner`, `bluetooth` et `netdev`.

Si `figatellix` crée un fichier, celui-ci lui appartiendra ainsi qu'à son groupe primaire :

```
figatellix@maximum:~$ touch toto
figatellix@maximum:~$ ls -l toto
-rw-r--r-- 1 figatellix figatellix 0 avril 25 18:22 toto
```

De la même façon qu'un utilisateur peut être local ou avoir une portée sur tout le réseau, les groupes peuvent être locaux ou être définis par un service d'annuaire sur le réseau.

2.3.2 Fichiers relatifs aux groupes d'utilisateurs

Les informations relatives aux groupes locaux sont stockées dans deux fichiers : `/etc/group` et `/etc/gshadow`. Par analogie avec les fichiers relatifs aux utilisateurs, ces deux fichiers appartiennent au super-utilisateur ; `/etc/group` est lisible par tout le monde, tandis que `/etc/gshadow` n'est lisible par le super-utilisateur et les membres du groupe shadow.

```
coti@maximum:~$ ls -l /etc/gshadow /etc/group
-rw-r--r-- 1 root root   925 mars  20 17:37 /etc/group
-rw-r----- 1 root shadow 777 mars  20 17:37 /etc/gshadow
```

Le fichier `/etc/group` contient les définitions des groupes, c'est-à-dire leur noms, leur gid et les utilisateurs qui en font partie. Il est constitué d'une ligne par groupe et chaque ligne contient quatre champs séparés par le symbole " :" :

- `groupname` : le nom du groupe ;
- `passwd` : le mot de passe du groupe, non-affichable (en réalité on voit un x, comme dans le fichier `/etc/passwd`) ;
- `GID` : le numéro du groupe
- users : la liste des utilisateurs faisant partie du groupe en tant que groupe secondaire et séparés par des virgules, ce champ pouvant rester vide si aucun utilisateur n'appartient à ce groupe.

Il est à noter que seuls les utilisateurs dont c'est le groupe secondaire sont listés dans les groupes du fichier `/etc/group`. Le groupe primaire de chaque utilisateur est quant à lui défini dans le fichier `/etc/passwd`.

Par exemple, voici un extrait d'un fichier `/etc/group` :

```
root:x:0:
cdrom:x:24:figatellix
audio:x:29:pulse,figatellix
figatellix:x:1001:
```

On voit qu'aucun utilisateur n'appartient au groupe `root` en tant que groupe secondaire. Ce groupe est le groupe primaire du super-utilisateur (par conséquent non listé ici). C'est également le cas du groupe `figatellix`, qui n'est groupe secondaire d'aucun utilisateur.

L'utilisateur `figatellix` a (entre autres) pour groupes secondaires les groupes `cdrom` et `audio`. L'utilisateur `pulse` appartient lui aussi au groupe `audio`.

Le fichier `/etc/gshadow` contient les mots de passe des groupes. Il est constitué d'une ligne par groupe et chaque ligne contient quatre champs séparés par le symbole " :" :

- `groupname` : le nom du groupe ;
- `passwd` : le mot de passe du groupe, crypté, comme dans le fichier `/etc/shadow`) ;
- `group admin` : la liste des utilisateurs définis comme administrateurs de ce groupe, séparés par des virgules ;
- `group members` : la liste des utilisateurs faisant partie du groupe en tant que groupe secondaire et séparés par des virgules.

Les deux derniers champs peuvent rester vides. De façon analogue au contenu du fichier `/etc/shadow`, les groupes contenant un astérisque ("*") à la place du mot de passe ne peuvent être rejoints au moyen d'un mot de passe. C'est en réalité le cas dans la majeure partie des cas. Les groupes ayant un point d'exclamation (" !") ne peuvent eux être rejoints que par les membres de ce groupe.

Les administrateurs d'un groupe sont autorisés à ajouter ou à supprimer des membres dans le groupe en utilisant la commande `gpasswd`.

Par exemple, voici un extrait d'un fichier `/etc/gshadow` :

```
root:*::
cdrom:*::figatellix
audio:*::pulse,figatellix
figatellix:!::
```

On voit ici que les groupes `root`, `cdrom` et `audio` ne peuvent pas être rejoints en utilisant un mot de passe. Le groupe `figatellix` peut quant à lui l'être, mais uniquement par les utilisateurs en faisant partie. Dans ces groupes, aucun utilisateur n'est administrateur.

2.4 Création d'un utilisateur

Un utilisateur est ajouté au système (c'est-à-dire créé) avec la commande `adduser`. Elle permet de créer un utilisateur local comme un utilisateur réseau (NIS, LDAP ou autre). De façon évidente, seul le super-utilisateur a la possibilité de créer un utilisateur.

Le comportement précis de cette commande est en partie spécifique à la version d'Unix que vous utilisez et à sa distribution. Par exemple, certaines distributions Linux demandent de saisir un mot de passe pour l'utilisateur lors de sa création, tandis que d'autres le créent sans lui affecter de mot de passe, et vous devez ultérieurement en créer un avec la commande `passwd`.

La commande `useradd` permet elle aussi de créer un utilisateur, mais celui-ci est forcément local. Elle ne permet pas de créer d'utilisateur réseau.

Ces commandes permettent notamment de spécifier le shell de connexion, le répertoire utilisateur, l'expiration éventuelle du compte et les groupes auxquels l'utilisateur doit appartenir.

2.4.1 Configuration de la création d'utilisateurs

Les valeurs utilisées lors de la création d'un utilisateur sont définies dans le fichier `/etc/login.defs` et dans `/etc/adduser.conf`. Le premier définit la configuration de la commande `useradd`, tandis que le second définit la configuration de la commande `adduser`.

On y trouve notamment des valeurs intéressantes comme les bornes de l'intervalle d'iud et gid utilisables (`FIRST_UID` et `LAST_UID` pour les uid, `FIRST_GID` et `LAST_GID` pour les gid). On a aussi `FIRST_SYSTEM_UID` et `FIRST_SYSTEM_GID`.

2.4.2 Profil par défaut

Lorsqu'un utilisateur est créé, on peut créer en même temps son répertoire utilisateur. Celui-ci est alors rempli avec un *squelette* : on copie dedans le contenu du répertoire `/etc/skel` (ou tout autre répertoire passé en paramètre de la création de l'utilisateur). Par exemple, si l'on souhaite que les utilisateurs soient créés avec un répertoire utilisateur contenant un répertoire `documents` et un fichier `hello`, il faut créer ce répertoire et ce fichier dans le répertoire `/etc/skel`.

2.4.3 Fichiers modifiés

Lorsqu'un utilisateur est créé, une ligne est ajoutée dans les fichiers `/etc/passwd` et `/etc/shadow`. De plus, on crée souvent en même temps un groupe portant le même nom que l'utilisateur : par conséquent, les fichiers `/etc/group` et `/etc/gshadow` sont modifiés pour ajouter ces groupes. Le nouvel utilisateur peut aussi être inséré dans des groupes secondaires : les fichiers `/etc/group` et `/etc/gshadow` sont alors modifiés ici encore.

2.5 Manipulation de groupes

Il est possible de manipuler les groupes en y ajoutant et en y retirant des utilisateurs. Deux types d'utilisateurs ont la possibilité d'effectuer ces opérations : le super-utilisateur, comme pour toutes les tâches d'administration, et les administrateurs du groupe concerné.

Le super-utilisateur peut manipuler les groupes de deux façons : soit en utilisant des commandes spécifiques, soit en éditant directement les fichiers concernés.

Par exemple, on peut ajouter un utilisateurs dans un groupe (qui est alors son groupe secondaire) en éditant le fichier `/etc/group` pour ajouter son login dans le dernier champ de la ligne correspondant à ce groupe. On peut aussi utiliser la commande `usermod` avec les options `-a -G` :

```
root@maximum:~# usermod  -a -G backup figatellix
```

La commande `groupmod` permet quant à elle de modifier le groupe lui-même : son gid, son mot de passe ou son nom.

3 Fixer des limites : permissions et limitations

3.1 Permissions associées aux fichiers

Les droits d'accès, ou les permissions, définissent ce que les utilisateurs peuvent faire avec un fichier donné. On définit, relativement à un fichier, trois catégories d'utilisateurs :

— L'utilisateur propriétaire du fichier
— Le groupe propriétaire du fichier
— Le reste du monde

Sur chaque fichier, on définit des permissions pour chaque catégorie d'utilisateurs. On peut définir trois permissions :

— Lecture
— Écriture
— Exécution

S'il s'agit d'un répertoire, sa lecture est le fait de lister les fichiers qu'il contient, l'écriture est le fait de créer des fichiers dedans, d'en supprimer ou de modifier leurs propriétés et l'exécution est la traversée de ce répertoire.

La commande `ls -l` nous informe sur les droits associés à un fichier :

```
coti@maximum:~/repertoire$ ls -l
total 16
lrwxrwxrwx 1 coti users   27 avril 27 16:48 lien -> /boot/vmlinuz-3.2.0-4-amd64
-rwxr-xr-x 1 coti users 5837 avril 27 16:48 plante
-rw-r--r-- 1 coti users  225 avril 27 16:48 plante.c
drwxr-xr-x 2 coti users 4096 avril 27 16:48 subdir
```

La première colonne nous donne le *mode* de chaque fichier. Le mode est composé de dix caractères : un indicateur de type (un caractère) puis les droits d'accès à ce fichier (neuf caractères).

Le premier caractère du mode est l'indicateur qui indique de quel type de fichier il s'agit. Il peut être :

— `d` : un répertoire (directory)

— `b` : un fichier spécial en mode bloc
— `c` : un fichier spécial en mode caractère
— `l` : un lien symbolique
— `p` : un pipeline Unix
— `s` : une socket
— `-` : un fichier normal

Dans l'exemple ci-dessus, on voit que le premier fichier (appelé `lien`) est un lien symbolique, les deux suivants sont des fichiers ordinaires et le dernier est un répertoire.

Les droits associés au fichier sont sur neufs caractères. En réalité, il s'agit de trois blocs de trois caractères :

— Les droits de l'utilisateur propriétaire du fichier
— Les droits du groupe propriétaire du fichier
— Les droits du reste du monde

Chaque bloc est composé de trois caractères, qui indiquent respectivement les droits en exécution, en écriture et en lecture. Si la permission est présente, il y a la lettre correspondante (`x`, `w` ou `r`). Sinon, il y a un tiret (`-`).

Par exemple, on a :

```
coti@maximum:~/repertoire$ ls -l plante
-rwxr-xr-x 1 coti users 5837 avril 27 16:48 plante
```

Les droits associés au fichier ci-dessus sont les suivants :

— Le propriétaire du fichier a le droit d'exécuter, écrire et lire le fichier (droits `rwx`)
— Les membres du groupe propriétaire du fichier ont le droit de l'exécuter et de le lire mais pas d'écrire dessus (droits `r-x`)
— Le reste du monde a le droit de l'exécuter et de le lire mais pas d'écrire dessus (droits `r-x`)

Attention, il est à noter que dans le cas d'un lien symbolique, ces droits s'appliquent au lien lui-même et non pas au fichier pointé par ce lien. Par exemple, dans le répertoire que nous examinons ici, le symbolique a des permissions très différentes de celles du fichier sur lequel il pointe :

```
coti@maximum:~/repertoire$ ls -l lien
lrwxrwxrwx 1 coti users 27 avril 27 16:48 lien -> /boot/vmlinuz-3.2.0-4-amd64
coti@maximum:~/repertoire$ ls -l /boot/vmlinuz-3.2.0-4-amd64
-rw-r--r-- 1 root root 2837536 févr.  2 01:51 /boot/vmlinuz-3.2.0-4-amd64
```

Les permissions d'un fichier peuvent être changées avec la commande `chmod`. On peut lui passer soit les permissions à appliquer au fichier, soit en ajouter ou en supprimer.

Par exemple, si l'on veut appliquer au fichier `plante.c` toutes les permissions pour son utilisateur propriétaire et les droits en lecture et en exécution pour son groupe propriétaire et le reste du monde, on va lui appliquer les droits `rwxrxrx` :

```
coti@maximum:~/repertoire$ chmod =rwxrxrx plante.c
coti@maximum:~/repertoire$ ls -l plante.c
-rwxr-xr-x 1 coti users 225 avril 27 16:48 plante.c
```

On peut aussi ajouter ou supprimer une permission à une catégorie d'utilisateurs en particulier. Dans ce cas, on désigne cette catégorie par une lettre : `o` pour le propriétaire (owner), `g` pour le groupe propriétaire et `a` pour le reste du monde (all). On ajoute un droit en particulier avec un signe `+` et on supprime avec le signe `-`.

```
coti@maximum:~/repertoire$ chmod g-x plante.c
coti@maximum:~/repertoire$ ls -l plante.c
-rwxr--r-x 1 coti users 225 avril 27 16:48 plante.c
coti@maximum:~/repertoire$ chmod g+w plante.c
coti@maximum:~/repertoire$ ls -l plante.c
-rwxrw-r-x 1 coti users 225 avril 27 16:48 plante.c
```

Les permissions peuvent aussi être exprimées sous forme numérique. Les trois caractères relevant de chaque catégorie d'utilisateurs sont vus chacun comme des bits formant un nombre : les deux caractères sont donc exprimés comme trois nombres de 3 bits. Lorsque le droit est présent, on met ce bit à 1 ; dans le cas contraire, on le met à 0.

Par exemple, les droits `rwxr-x-x` deviennent en binaire `111 101 001`, soit, en décimal, 751. On peut passer cette forme numérique à `chmod` pour appliquer des permissions sur un fichier :

```
coti@maximum:~/repertoire$ chmod 751 plante.c
coti@maximum:~/repertoire$ ls -l plante.c
-rwxr-x--x 1 coti users 225 avril 27 16:48 plante.c
coti@maximum:~/repertoire$ chmod 644 plante.c
coti@maximum:~/repertoire$ ls -l plante.c
-rw-r--r-- 1 coti users 225 avril 27 16:48 plante.c
```

3.2 Propriétaire d'un fichier

Lorsqu'un fichier est créé, il a pour utilisateur propriétaire celui qui l'a créé et pour groupe propriétaire le groupe primaire de cet utilisateur. Il est possible de changer ce propriétaire avec la commande `chown` (pour *change owner*).

Par exemple, si l'ont crée un fichier avec touch et que l'on regarde quels sont son utilisateur et son groupe propriétaires, on voit qu'ils sont ceux de l'utilisateur qui l'ont créé :

```
coti@maximum:/tmp$ touch toto
coti@maximum:/tmp$ ls -l toto
-rw-r--r-- 1 coti 1000 331 avril 27 16:39 toto
```

Seul le propriétaire a les droits en écriture sur ce fichier. Par conséquent, même lui-même n'a pas le droit d'en changer le propriétaire. Il faut les privilèges du super-utilisateur pour le faire. La commande `chown` permet de changer soit l'utilisateur propriétaire uniquement, soit l'utilisateur et le groupe propriétaires en même temps.

```
root@maximum:/tmp# chown figatellix toto
root@maximum:/tmp# ls -l toto
-rw-r--r-- 1 figatellix 1000 331 avril 27 16:39 toto
root@maximum:/tmp# chown figatellix:audio toto
root@maximum:/tmp# ls -l toto
-rw-r--r-- 1 figatellix audio 331 avril 27 16:39 toto
```

3.3 Autres approches

Ce mécanisme de permissions est le mécanisme standard d'Unix. Ils ont des limitations assez fortes du fait de leur caractère assez simple et basique. Par exemple, il n'est pas possible de donner explicitement des droits à plusieurs utilisateurs. La seule solution, dans ce cas, est de mettre ces utilisateurs dans un groupe et de définir des permissions pour ce groupe. Il n'est pas possible, par exemple, de donner des droits en écriture à deux utilisateurs.

Pour palier à cette limitation, d'autres mécanismes existent, parmi lesquels les Access Control Lists (ACL). Le système de fichiers AndrewFS implémente un système d'ACL qui permet de définir une liste d'utilisateurs ayant des droits donnés sur un fichier.

3.4 Permissions par défaut

Lorsqu'un fichier est créé par un utilisateur, des droits lui sont associés dès sa création. Ces droits sont définis par le *masque de création* de l'utilisateur. La commande `umask` permet de connaître et de définir ce masque.

Il est composé de quatre chiffres : le premier correspond à l'indicateur de type de fichiers (on n'y touche pas, c'est donc 0) et les trois suivants aux permissions. Les permissions sont définies en retranchant le masque d'une valeur : 0777 pour les répertoires et 0666 pour les fichiers. Pour connaître son umask :

```
coti@maximum:~/repertoire$ umask
0022
```

Il est ici de 0022. Cela signifie que lorsqu'un répertoire est créé, les droits qui lui sont affectés sont calculés par l'opération $(0777 - 0022) = 755$:

```
coti@maximum:~/repertoire$ mkdir titi
coti@maximum:~/repertoire$ ls -l
drwxr-xr-x 2 coti users 4096 avril 27 17:18 titi
```

De même, lorsqu'un fichier est créé, les droits qui lui sont affectés sont calculés par l'opération $(0666 - 0022) = 644$:

```
coti@maximum:~/repertoire$ touch toto
coti@maximum:~/repertoire$ ls -l toto
-rw-r--r-- 1 coti users 0 avril 27 17:18 toto
```

Le umask peut être modifié :

```
coti@maximum:~/repertoire$ umask 0000
coti@maximum:~/repertoire$ touch toto2
coti@maximum:~/repertoire$ ls -l toto2
-rw-rw-rw- 1 coti users 0 avril 27 17:28 toto2
```

Dans l'exemple ci-dessus, on met le umask à 0000. Lorsqu'un fichier est créé, ses droits sont donc $(0666 - 0000) = 0666$, soit `rw-rw-rw-`.

3.5 Obtenir des droits d'administration

Il existe deux façons d'obtenir (de façon légitime) des droits d'administrateur. La première consiste à changer d'identité pour devenir ce super-utilisateur :

```
coti@maximum:~$ su -
Mot de passe :
root@maximum:~# id
uid=0(root) gid=0(root) groupes=0(root)
```

Cependant, cette approche nécessite de connaître le mot de passe du super-utilisateur. Si plusieurs utilisateurs d'un système ont besoin d'effectuer des tâches réservées au super-utilisateur, l'administrateur système n'a pas forcément envie de leur donner à tous le mot de passe. De plus, elle permet d'avoir accès à ces privilèges d'administration sur toute la machine et de façon incontrôlée.

La deuxième façon de faire est de rester sous son identité d'utilisateur normal mais d'obtenir temporairement des privilèges de super-utilisateur. C'est ce à quoi sert la commande `sudo`.

Un des avantages de `sudo` est que les privilèges du super-utilisateur ne sont accordés que temporairement à l'utilisateur, uniquement le temps d'exécuter sa commande. Ainsi, celui-ci ne risque pas d'oublier de se déconnecter de l'identité du super-utilisateur et de commettre des erreurs ultérieures en ayant trop de privilèges.

```
coti@maximum:~$ head /var/log/syslog
head: impossible d'ouvrir « /var/log/syslog » en lecture: Permission non accordée
coti@maximum:~$ sudo head /var/log/syslog
[sudo] password for coti:
Apr 27 07:35:12 maximum rsyslogd: [origin software="rsyslogd" swVersion="5.8.11"
x-pid="2200" x-info="http://www.rsyslog.com"] rsyslogd was HUPed
Apr 27 07:35:13 maximum anacron[9077]: Job ‘cron.daily’ terminated
```

Les actions effectuées par les utilisateurs avec `sudo` sont journalisées dans le fichier `/var/log/auth.log`. Ainsi, il est possible de garder une trace de ce que les utilisateurs ont fait avec leurs privilèges temporaires.

```
coti@maximum:~$ sudo tail /var/log/auth.log
[...]
Apr 27 17:39:10 maximum sudo:     coti : TTY=pts/3 ; PWD=/users/coti ; USER=root
; COMMAND=/usr/bin/head -n 2 /var/log/syslog
[...]
```

Dans l'exemple ci-dessus, l'utilisateur doit taper son mot de passe pour obtenir des privilèges avec `sudo`. Généralement, il n'est pas demandé à nouveau pour utiliser `sudo` pendant une certaine période après avoir été tapé. Ce comportement peut être configuré : il est possible de ne pas demander de mot de passe pour utiliser `sudo`. Cependant, pour des raisons de sécurité, il vaut mieux demander de le taper.

Le comportement de `sudo`, les utilisateurs ayant le droit de l'utiliser et dans quelles conditions, est défini dans le fichier `/etc/sudoers`. Ce fichier s'édite avec la commande `visudo`. On y définit quels groupes et quels utilisateurs ont le droit d'utiliser `sudo` :

```
coti@maximum:~$ sudo cat /etc/sudoers
[...]

# User privilege specification
root    ALL=(ALL:ALL) ALL
figatellix      ALL=(ALL:ALL) ALL
coti            ALL=(ALL:ALL) ALL
[...]
```

Dans l'exemple ci-dessus, on a trois utilisateurs qui ont le droit de tout faire avec `sudo`. On peut limiter à quelques commandes listées après `ALL`. On peut aussi ajouter des options, comme `NOPASSWD` pour ne pas demander à l'utilisateur de fournir son mot de passe. Par exemple :

```
coti@maximum:~$ sudo cat /etc/sudoers
[...]
figatellix      ALL = NOPASSWD: /bin/ls
[...]
```

Dans l'exemple ci-dessus, l'utilisateur `figatellix` n'a le droit d'utiliser que la commande `/bin/ls` avec `sudo`, et son mot de passe ne lui sera pas demandé.

```
figatellix@maximum:~$ sudo ls
Bureau  Documents  Images  Modèles  Musique
figatellix@maximum:~$ sudo mkdir toto
[sudo] password for figatellix:
Sorry, user figatellix is not allowed to execute '/bin/mkdir toto' as root on maximum.lipn.univ-paris13.fr.
```

Ci-dessus, on voit que cet utilisateur a bien pu utiliser `/bin/ls` avec `sudo`, mais pas `mkdir` car cela ne lui a pas été autorisé dans le fichier `/etc/sudoers`. On peut voir ce refus journalisé dans le fichier `/var/log/auth.log`.

```
Apr 27 17:51:59 maximum sudo: figatellix : command not allowed ; TTY=pts/3 ;
PWD=/home/figatellix ;
USER=root ; COMMAND=/bin/mkdir toto
```

Attention : si l'utilisation de `sudo` est plus sure que le fait d'être super-utilisateur, cette commande, configurée de façon trop laxiste, permet cependant de devenir super-utilisateur, par exemple avec `sudo su -` (permet d'exécuter la commande `su -` avec les privilèges du super-utilisateur, ne demandant donc pas le mot de passe du super-utilisateur) ou `sudo bash` (exécute un interpréteur de commandes en tant que super-utilisateur).

3.6 Limitation d'espace : les quotas

Il peut être utile de limiter l'espace pris par un répertoire sur un système de fichiers. Par exemple, dans un système multi-utilisateur, il est courant de limiter l'espace que peuvent prendre les répertoires personnels des utilisateurs afin d'éviter qu'un utilisateur prenne trop de place sur le système de fichiers.

Les systèmes POSIX comme Unix proposent un système de limitation de l'espace par des *quotas*. Activés sur une partition, ils permettent de limiter l'espace utilisé par les utilisateurs sur son système de fichiers. Par exemple, avec un quota de 50 Mo, un utilisateur ne pourra pas utiliser plus de 50 Mo sur toute la partition.

3.6.1 Les limites fixées par quotas

Les quotas définissent deux limites :
— La *limite dure* (hard limit), qui ne peut pas être dépassée
— La *limite douce* (soft limit), qui peut être dépassée temporairement
Après une certaine période, dire *période de grâce*, la limite douce se transforme en limite dure et il est impossible d'augmenter la taille de l'espace utilisé sur la partition.

Attention, lorsqu'un utilisateur est en limite dure et qu'il n'est plus possible d'augmenter l'espace qu'il utilise, il devient compliqué de se connecter à son compte. En effet, les gestionnaires de fenêtres ont besoin d'utiliser des fichiers sur le répertoire personnel au moment de l'ouverture de session (connexion à un système via le gestionnaire de fenêtres). Quand un utilisateur est en limite dure, il lui devient donc impossible de se connecter en utilisant le gestionnaire de fenêtres. Il convient donc de ne pas prendre cette limite douce et la période de grâce à la légère.

Les quotas s'appliquent sur deux grandeurs :
— L'espace utilisé par un utilisateur ;
— Le nombre d'inodes, c'est-à-dire ne nombre de fichiers qu'a cet utilisateur.

On a donc quatre limites qui sont définies : la limite dure et la limite souple pour l'espace ainsi que la limite dure et la limite souple pour le nombre d'inodes.

3.6.2 Activation des quotas sur une partition

Dans le fichier `/etc/fstab`, il faut les activer explicitement sur la partition concernée. Il existe deux sortes de quotas : les quotas de groupe (définissant un quota pour un groupe d'utilisateurs) et les quotas d'utilisateurs (définissant un quota par utilisateur). On passe les options `usrquota` et/ou `grpquota` pour activer respectivement les quotas d'utilisateurs et de groupe sur une partition donnée :

```
/dev/sdb1 /home ext3 defaults,usrquota,grpquota  1 1
```

Lorsque la partition sera montée, ces options seront prises en compte. Elles ne sont évidemment pas prises en compte immédiatement après l'édition du fichier : il faut remonter la partition. Pour la remonter à chaud :

```
coti@maximum:~$ sudo mount -o remount /home
```

Les quotas sont configurés dans deux fichiers situés à la racine de la partition concernée. Par exemple, si l'on souhaite définir des quotas sur la partition montée au point de montage `/home`, ces fichiers doivent être situés directement dans `/home`, qui est la racine de cette partition (rappel : on monte la racine d'une partition à son point de montage dans l'arborescence de fichiers).

Les quotas d'utilisateurs sont définis dans le fichier `quota.user` et les quotas de groupe sont définis dans `quota.group`. Ces deux fichiers doivent avoir les permissions 600.

Les quotas s'activent avec le commandes `quotaon` et `quotaoff`. Une fois activés, ils doivent être mis en place avec la commande `quotacheck`.

```
coti@maximum:~$ sudo quotaon -a
coti@maximum:~$ sudo quotacheck -vguma -F vfsv0
coti@maximum:~$ sudo quotaoff
```

3.6.3 Configuration des quotas

La commande `edquota` sert à éditer les quotas sur une partition donnée. Elle ouvre un éditeur de texte dans lequel il est possible de définir les limites. Elle peut s'utiliser pour un utilisateur (option `-u`) ou pour un groupe (option `-g`).

```
coti@maximum:~$ sudo edquota -u user1
Quotas disque pour user user1 (uid 1005) :
 Système de fichiers          blocs       souple     stricte   inodes    soupl$
  /dev/sdb5                      24        150000   0         13        0$
```

Avec la commande `repquota`, on peut voir les définitions des quotas sur une partition donnée. Cette commande affiche un tableau récapitulatif des quotas et de l'état de l'utilisation de l'espace sur la partition concernée.

```
coti@maximum:~$ sudo repquota /home
*** Rapport pour les quotas user sur le périphérique /dev/sdb5
Période de sursis bloc : 7days ; période de sursis inode : 7days
                        Block limits                File limits
Utilisateur     utilisé souple stricte sursis utilisé souple stricte sursis
------------------------------------------------------------------------
root      --      20       0       0              2     0     0
user1     --      24     150       0             13     0     0
user2     --      16       0       0              4     0     0
```

On peut aussi n'afficher que l'état d'utilisation d'un utilisateur, avec la commande `quota -u` :

```
coti@maximum:~$ sudo quota -u user1
Disk quotas for user user1 (uid 1005):
Système fichiers   blocs   quota  limite  sursisfichiers   quota  limite  sursis
      /dev/sdb5      24     150       0              13       0       0
```

3.7 Limites système

Le système d'exploitation permet lui aussi de fixer des limites aux utilisateurs. À l'inverse des quotas décrits dans la section 3.6 et qui sont fixés au niveau d'un système de fichiers et n'ont de portée que sur celui-ci, les limites sont fixées par le système d'exploitation et ont portée sur toute la machine.

3.7.1 Limites fixées par le système

Les limites peuvent être fixées dans deux buts :

— Pour l'utilisateur lui-même, comme protection ou pour ne pas voir certaines choses. Par exemple, quand il programme plante, le système peut créer un fichier contenant l'état de sa mémoire au moment où il a planté (core dump). En fixant la limite de la taille de ce fichier à zéro, on signifie au système que l'on ne souhaite pas qu'il le fasse.

— Par l'administrateur, pour évier qu'un utilisateur utilise trop de ressources sur le système.

Ainsi, on a deux sortes de limites :

— La *limite douce* (soft limit), fixée par l'utilisateur ;

— La *limite dure* (hard limit), fixée par l'administrateur.

L'utilisateur peut modifier sa limite douce jusqu'à atteindre la limite dure. Seul l'administrateur peut augmenter la limite dure, mais l'utilisateur peut la diminuer.

Avec la commande `ulimit`, on peut voir les limites fixées pour un utilisateur. L'option `-a` permet de les voir toutes :

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 63602
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 63602
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

On voit ci-dessus, entre autres, que pour l'utilisateur `figatellix`, la taille du fichier de core dump est limitée à 0 (donc on ne produit pas de tel fichier), qu'il peut ouvrir au plus 1024 fichiers simultanément et avoir au plus 63602 processus exécutés en même temps.

3.7.2 Configuration des limites système

Les limites système sont définies dans le fichier `/etc/security/limits.conf`. Pour chaque utilisateur ou groupe concerné, on trouve une ligne qui spécifie :

— le nom de l'utilisateur ou du groupe (dans ce cas, on le précède d'un signe)
— le type de limite : `hard` ou `soft`
— l'élément concerné par la limite : par exemple `nproc` pour le nombre de processus, `fsize` pour la taille maximale de fichier que l'utilisateur est autorisé à créer, `stack` pour la taille maximale de la pile...
— la valeur de cette limite

Par exemple, on peut définir des limites dure et douce pour le nombre de processus exécutés pour un moment du groupe `etudiants` de la façon suivante :

```
@etudiants     hard     nprocs     1024
@etudiants     soft     nprocs     512
```

Ces valeurs peuvent également être modifiées par la commande `ulimit`. La limite douce est spécifiée avec l'option `-S` et la limite dure est spécifiée avec l'option `-H`. La limite elle-même est spécifiée avec une option spécifique à chacune, qui peut être vue dans la sortie de `ulimit -a`. Par exemple, l'option `-n` correspond au nombre de fichiers ouverts.

```
coti@maximum:~$ ulimit -n
1024
coti@maximum:~$ ulimit -n 2086
coti@maximum:~$ ulimit -n
2086
```

4 Le stockage sur disque

4.1 Un peu d'architecture matérielle : la mémoire

Un ordinateur a pour but de manipuler des données. C'est pourquoi le stockage et l'accès à ces données est un élément des plus critiques dans sa conception. Un ordinateur dispose en réalité d'un grand nombre d'unités de stockage de mémoire, différant entre elles par leur taille, leur volatilité et leur rapidité d'accès.

La mémoire la plus rapide est située dans le processeur lui-même : ce sont les *caches*. Il en existe plusieurs, et de plusieurs niveaux. Ils sont effacés très souvent, simplement en chargeant et déchargeant des données nécessaire à l'exécution de l'application.

Le plus petit et le plus rapide, situé dans chaque cœur du processeur, est le cache de niveau 1, ou *cache L1*. Il existe en réalité deux caches L1 : le cache de données et le cache d'instructions. C'est la mémoire la plus proche de l'Unité Arithmétique et Logique (ALU) qui effectue les calculs eux-mêmes. Le cache L1 est généralement très petit (32 Ko sur un processeur Intel Core i7-2600), mais très rapide d'accès. Également situé dans le cœur mais plus éloigné de l'ALU, on trouve le cache de niveau 2, ou *cache L2*. Il est plus gros (256 Ko sur un processeur Intel Core i7-2600) et un peu plus lent. Enfin, sur chaque CPU, on trouve un cache de niveau 3, ou *cache L3*, qui est partagé entre les cœurs. Il est plus gros (8 Mo sur un processeur Intel Core i7-2600) mais plus lent que le cache L2, et il est partagé entre les cœurs.

Les données nécessaires à l'exécution des applications ne tenant pas en cache sont mises dans la *mémoire vive* de l'ordinateur, communément appelée RAM par abus de langage. La mémoire vive est beaucoup plus lente que les caches, mais également beaucoup plus grosse : de l'ordre de 4 à 16 Go sur les machines de bureau, jusqu'à 512 Go à 1 To sur des machines de calcul et plusieurs To sur certains serveurs spécifiques, comme des serveurs de base de données in-memory. La mémoire vive est effacée à chaque mise hors tension. Ainsi, lorsque la machine est éteinte, les données contenues dans sa mémoire vive sont perdues.

La seule mémoire qui ne soit pas volatile sur un ordinateur (hors stockage externe) est son *disque dur*. On n'utilise le disque dur pour stocker des données des applications en cours d'exécution que dans des cas très particuliers : lorsque la mémoire vive est pleine, en utilisant un mécanisme de swapping qui décharge temporairement des données sur une partition spécifique du disque dur (la partition *swap*) ou lorsque l'on effectue des calculs out-of-core qui ne tiennent pas en mémoire vive et qui stockent temporairement des données sur le disque dur. Dans tous les autres cas, les données stockées sur le disque dur sont censées être *non-volatiles* et ne jamais être effacées, en dehors des effacements explicites voulus par l'utilisateur ou l'administrateur.

4.2 Les disques locaux

4.2.1 Types de disques

Du point de vue du système d'exploitation, un disque local diffère selon la technologie de stockage utilisée et selon la façon dont il est connecté à l'ordinateur.

Type de stockage

La technologie historique pour les disques durs actuels (qui est le type de disque réellement désigné par l'expression "disque dur") est mécanique. Un ensemble de plateaux tournent autour d'un axe et des têtes de lecture se déplacent sur ces plateaux. C'est extrêmement fragile et lent : la tête de lecture doit physiquement se déplacer pour aller lire des données situées loin de l'endroit où elle vient de lire. De plus, leur fragilité en fait l'élément le plus vulnérable aux pannes d'un ordinateur. Cette technologie date des années 50 et est actuellement toujours utilisée dans les ordinateurs.

Une technologie plus récente de stockage non volatile n'utilisant aucun élément mécanique mais uniquement de la mémoire flash est appelée SSD : Solid State Device. Les disques SSD sont beaucoup plus robustes et plus rapides que les disques durs mécaniques. Ils consomment aussi beaucoup moins d'énergie. Cependant, les cellules elles-mêmes ne sont pas très fiables et ont une durée de vie limitée. Ainsi, la capacité de stockage d'un disque SSD diminue au cours de son utilisation. De plus, ils sont actuellement trop petits et chers pour remplacer totalement les disques durs mécaniques.

D'autres technologies plus anciennes comme les cartes perforées ou les bandes magnétiques sont aujourd'hui beaucoup moins répandues et réservées à des usages spécifiques.

Connectique

Dans les ordinateurs personnels, on utilisait jusqu'au milieu des années 2000 des interfaces IDE (ou PATA, pour Parallel ATA) pour connecter les disques durs. Physiquement, on utilisait une nappe de 40 ou 44 fils sur deux rangées (2x20 ou 2x22) permettant de transmettre 16 bits à la fois. Pour doubler

la vitesse de transmission, on a ensuite introduit des nappes de 80 fils. Les version les plus rapides permettaient de transmettre 66 Mo par seconde.

Utilisant des connecteurs plus petits et un débit plus rapide, la norme Serial ATA (ou SATA) est aujourd'hui la plus utilisée dans les ordinateurs personnels. Elle permet d'obtenir un débit de plusieurs Gbits par seconde (plusieurs centaines de Mo par seconde). Le débit de la liaison entre le disque et l'ordinateur est alors supérieur à celui offert par un disque dur mécanique (environ 150 Mo par seconde).

Certains serveurs utilisent l'ancienne norme SCSI (Small Computer System Interface), qui offre de meilleurs performances que IDE. Elle présente un intérêt moindre depuis le développement de SATA, et est aujourd'hui remplacée par la liaison Fibre Channel. On utilise cette dernière dans des serveurs de stockage utilisant le RAID ou dans des architectures SAN.

4.2.2 Représentation sous Linux

Un principe très important sous Unix est que *tout est fichier*. Les disques sont donc représentés comme des fichiers. On les trouve dans le répertoire `/dev` (pour "device").

Sous Linux, ils sont nommés de la façon suivante :

— La *première lettre* du nom du fichier indique le type de connectique utilisé : `h` pour une interface IDE, `s` pour une interface SATA ou SCSI.

— La *deuxième lettre* est un `d`.

— La *troisième lettre* sert à numéroter les disques : le premier utilise un `a`, le deuxième un `b`, etc...

Par exemple, le premier disque IDE sera représenté par le fichier `/dev/hda`. Le troisième disque SATA ou SCSI sera représenté par le fichier `/dev/sdc`.

Pour être utilisé, un disque doit être *partitionné*. Les partitions sont elles aussi vues par le système comme des fichiers. Leur nom commence par le nom du disque dont elles font partie, puis elles sont numérotées (à partir de 1) : `/dev/sda3` désigne la troisième partition du premier disque SATA ou SCSI.

Sous Mac OS (qui est une autre variante d'Unix), le système de nommage est légèrement différent :

— Le nom commence par `disk`

— Puis vient le numéro du disque, en commençant par 0

— Ensuite la lettre `s`

— Enfin le numéro de partition en commençant par 1

Par exemple, `/dev/disk1s2` désigne la deuxième partition du premier disque sur un système sous Mac OS X.

On peut utiliser un outil comme `fdisk` ou `(g)parted` pour partitionner un risque dur.

4.2.3 Systèmes de fichiers

Les disques de stockage (disque dur mécanique ou SSD) sont des dispositifs permettant de stocker des données sous forme d'un ensemble de blocs de données. Cependant, il faut que le système d'exploitation sache où trouver physiquement ces données sur le support de stockage. C'est le rôle du *système de fichiers* que d'organiser le stockage et l'accès des données sur une partition d'un disque.

Le système de fichiers fournit des fonctionnalités de base, comme l'ouverture et la fermeture d'un fichier. Mais c'est aussi à lui qu'incombe la gestion des permissions d'accès, par exemple.

On peut citer quelques systèmes de fichiers de stockage :

— FAT (puis FAT16, FAT32, VFAT...) : le système historique de MS DOS puis Windows jusqu'à Windows XP, et des cartes mémoire et clé USB. Très simple et fournissant peu de fonctionnalités (pas de permissions d'accès).

— NTFS est le système de fichiers actuellement utilisé par Windows. Il permet notamment d'utiliser des permissions d'accès aux fichiers.

— ext2, ext3, ext4 : utilisés par défaut par les systèmes Linux. Le système ext3 est principalement une évolution d'ext2 en lui ajoutant la journalisation des évènements, ext3 est une évolution d'ext3. Les trois systèmes de fichiers sont compatibles entre eux.

— btrfs est un système de fichiers "moderne" utilisant une structure de données algorithmique avancée pour stocker les index des fichiers.

— AndrewFS (AFS) est un système de fichiers distribué offrant des fonctionnalités de contrôle d'accès sur les fichiers plus avancées que les droits Unix standards.
— ZFS est un système de fichiers orienté pour les très gros volumes. Il est implémenté pour Solaris et NetBSD, mais pas totalement pour Linux.
— ReiserFS est le premier système de fichiers journalisé pour Linux. Il est particulièrement performant sur un grand nombre de fichiers et sur les petits fichiers.
— HFS et son évolution HFS+ est le système utilisé par Apple sous Mac OS X et dans ses iBidules (iPod, iPhone, iPad...).

Il existe d'autres systèmes de fichiers utilisés pour représenter des données qui ne sont pas liées à du stockage lui-même. Par exemple, tmpfs représente des données volatiles, ou procfs est utilisé en interne par le noyau pour représenter les processus en cours.

Sous Linux, on crée un système de fichiers avec la commande `mkfs`. L'option `-t` permet de spécifier le système de fichiers désiré.

Par exemple, la commande suivante permet de créer un système de fichier ext3 sur la cinquième partition du premier disque SATA ou SCSI :

```
root@maximum:~# mkfs -t ext3  /dev/sda5
```

On peut vérifier l'intégrité d'un système de fichiers ext2/ext3/ext4 avec la commande `e2fsck`. Évidemment, la partition vérifiée ne doit pas être montée dans l'arborescence de fichiers au moment où elle est vérifiée. Cet utilitaire est également utile pour réparer un système de fichiers journalisé qui a été arrêté brutalement sans que toutes les opérations aient été effectuées sur les données.

Un système de fichiers ext2/ext3/ext4 peut être redimensionnée avec l'utilitaire `resize2fs` : sa taille peut être réduite au augmentée. Attention cependant à ne pas trop la réduire : les données contenues doivent toujours tenir sur la nouvelle taille. Attention, `resize2fs` ne modifie pas la taille de la partition mais uniquement celle du système de fichiers.

4.2.4 Montage dans l'arborescence

Vous savez que les fichiers et les répertoires sont représentés dans les systèmes Unix sous forme d'une unique structure arborescente, représentée figure 1. Lorsqu'un système de fichiers est ajouté au système, il est *monté* à un endroit de cet arbre.

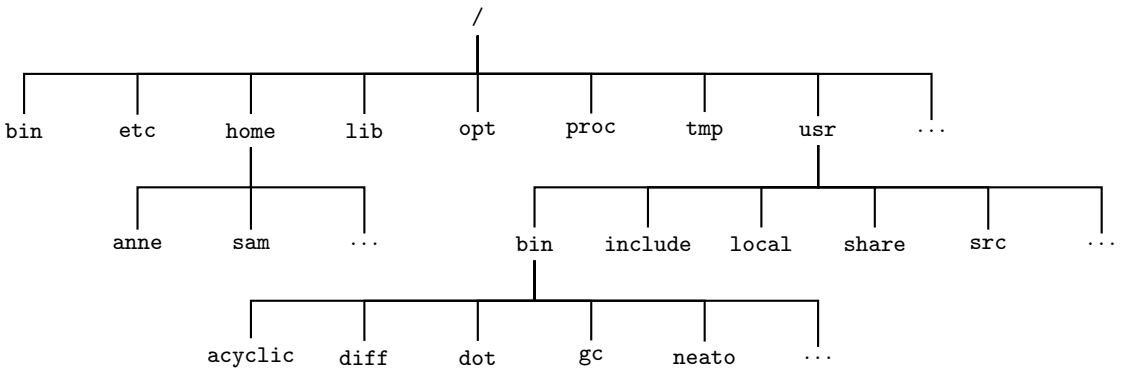


FIGURE 1 – Structure arborescente les fichiers sous Unix

Dans l'arborescence représentée figure 1, la racine (`/`) est sur une partition. Considérons que les répertoires utilisateurs sont situés sur une autre partition. Celle-ci est montée au *point de montage* `/home` de l'arborescence. La partition contient les répertoires `anne`, `sam`, etc, qui sont rattachés à l'arborescence au niveau de ce point de montage. Ainsi, on peut voir les contenus des diverses partitions utilisées dans une unique structure.

Pour voir les partitions montées sur le système, on utilise la commande `mount` :

```
coti@thorim:~$ mount
/dev/disk0s2 on / (hfs, local, journaled)
devfs on /dev (devfs, local, nobrowse)
map -hosts on /net (autofs, nosuid, automounted, nobrowse)
map auto_home on /home (autofs, automounted, nobrowse)
```

On voit ici que l'on se trouve sur un système sous Mac OS X où la partition `/dev/disk0s2` contient un système de fichiers HFS sur lequel se trouve la racine de l'arborescence de fichiers. Les autres systèmes de fichiers sont des systèmes de fichiers internes, utilisés par le système et non pas pour stocker des fichiers. Ils sont montés aux points `/dev`, `/net` et `/home`.

La commande `mount` sert également à monter un système de fichiers dans l'arborescence. Elle peut prendre quelques options, mais dans l'ensemble, elle s'utilise toujours de la façon suivante :

```
mount [options] <partition> <point de montage>
```

On lui donne d'abord la partition à monter (éventuellement en utilisant le nom du fichier correspondant, mais il peut s'agir d'une partition sur le réseau) puis le point de montage à utiliser. Par exemple, pour monter la partition `/dev/sdb5` (cinquième partition du deuxième disque SATA ou SCSI) au point de montage `/home` :

```
root@maximum:~# mount /dev/sdb5 /home
```

On peut supprimer cette partition de l'arborescence de fichiers en utilisant `umount` :

```
root@maximum:~# umount /home
```

On peut spécifier une fois pour toutes les fichiers à monter au démarrage dans le fichier `/etc/fstab`. Ce fichier contient une ligne par système de fichiers pouvant être monté, chaque ligne donnant le nom du fichier correspondant à la partition, le point de montage, le type de système de fichiers et quelques options. Si la partition n'est pas présente au démarrage, cela n'est pas grave : elle n'est juste pas montée, mais ce n'est pas bloquant.

Un exemple de fichier `/etc/fstab` :

```
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/dev/sda1 /               ext3    errors=remount-ro 0       1
/dev/sdb5 /home           ext3    defaults        0       2
/dev/sdb1 /opt            ext3    defaults        0       2
/dev/sda6 /usr            ext3    defaults        0       2
/dev/sda7 /var            ext3    defaults        0       2
/dev/sda5 none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
```

La première ligne commence par un `#` : il s'agit d'un commentaire, elle n'est pas prise en compte par le système.

Si la partition est connue du fichier `/etc/fstab`, on peut la monter simplement en donnant le nom du fichier la contenant ou le point de montage. Ainsi, avec le fichier `/etc/fstab` ci-dessus, les trois commandes suivantes sont équivalentes :

```
root@maximum:~# mount /dev/sdb5 /home
root@maximum:~# mount /dev/sdb5
root@maximum:~# mount /home
```

4.3 Stockage en réseau

Les fichiers stockés sur un disque dur ne sont accessibles directement que depuis cette machine. Dans un environnement de travail en réseau, cela présente l'inconvénient de lier les données à une machine. Par exemple, lorsque vous êtes en TP, vous ne revenez pas forcément sur la même machine, ni même dans la même salle, d'une séance à une autre. Il est donc extrêmement pratique de pouvoir accéder à des fichiers depuis n'importe quelle machine du réseau. Pour cela, on dispose de techniques de *stockage en réseau*.

4.3.1 NFS

NFS, ou Network File System, est comme son nom l'indique un système de fichiers en réseau. Il a été développé à l'origine par SUN, sa première version datant de 1984. Les version 1 et 2 utilisent UDP. La version 3 l'a étendu pour utiliser également TCP. La version 4 apporte un grand nombre de modifications, y compris dans son protocole.

Historiquement, NFS a été créé pour permettre à des machines sans disques d'accéder à des données situées sur un serveur central.

RPC NFS fonctionne en utilisant RPC. Le serveur de fichiers est un serveur RPC, et les requêtes des clients sont effectuées sous forme de requêtes RPC.

UDP ou TCP L'utilisation de UDP permet de réduire le traffic réseau en réduisant au maximum le nombre de messages protocolaires (pas de messages protocolaires avec UDP). Dans le cas de l'utilisation d'UDP, le serveur envoie au client en cookie, qui est une valeur aléatoire stockée du côté serveur et transmis en même temps que les requêtes RPC du client. Ce cookie permet au serveur d'autoriser le client à accéder au volume partagé.

Si le serveur redémarre, le cookie de chaque client reste valable : le redémarrage n'affecte donc pas les clients. Cependant, UDP étant un protocole sans état, les clients ne sont pas au courant de l'arrêt du serveur. Ils continuent donc de saturer le réseau de requêtes. C'est pourquoi TCP, plus résilient, est généralement préféré à UDP.

Authentification des clients NFS ne prévoit pas d'authentification des utilisateurs. L'autorisation d'accès au volume partagé est effectué sur les clients (les machines) et non pas sur les utilisateurs. NFS v4 permet un contrôle d'accès plus fin ; par ailleurs, les versions antérieures, et notamment le très répandu NFS v3 laissent accéder tous les utilisateurs d'un client autorisé.

Il faut aussi faire très attention au couplage avec NIS sur des machines sur lesquelles des utilisateurs peuvent être root. Sous l'identité du super-utilisateur, il est possible de prendre l'identité de n'importe quel utilisateur du système (local ou NIS) avec `su`. Une fois devenu l'autre utilisateur, on peut accéder à ses fichiers sur NFS.

Montage d'un volume NFS Le montage d'un volume NFS se fait comme le montage de n'importe quelle partition. Si l'on souhaite par exemple monter le répertoire `/users` d'un serveur `lipn-sfa` sur le point de montage `/home` de la machine locale :

```
root@maximum:~# mount lipn-sfa:/users /users
```

On peut utiliser des options avec l'option `-o`, comme par exemple `-o nosync` pour utiliser le mode asynchrone pour de meilleures performances sur un réseau qui manque de réactivité ou, au contraire, `-o sync` pour que les opérations soient synchrones entre le client et le serveur.

Lorsque l'on regarde les disques montés avec la commande `mount`, on voit également les volumes NFS apparaître :

```
coti@maximum:~$ mount
[...]
/dev/sdb5 on /home type ext3 (rw,relatime,errors=continue,user_xattr,acl,barrier=1,data=ordered,usrquota,
grpquota)
/dev/sda7 on /opt type ext3 (rw,relatime,errors=continue,user_xattr,acl,barrier=1,data=ordered)
lipn-sfa:/export/vol01/coti on /users/coti type nfs4(rw,noatime,vers=4.0,rsize=1048576,wsize=1048576,
namlen=255,hard,proto=tcp,port=0,timeo=600,retrans=2,sec=sys,clientaddr=10.10.0.217,local_lock=none,
addr=10.10.0.191)
```

On voit ici deux partitions de disques locaux montées (`/dev/sdb5` et `/dev/sda7`) ainsi qu'un répertoire NFS (`/export/vol01/coti` sur la machine `lipn-sfa`).

Système de fichiers côté serveur NFS est en réalité un protocole de communications sur le réseau, au-dessus de RPC. Ce n'est pas un système de fichiers. Côté serveur, le choix est laissé libre du système de fichiers utilisé localement pour le volume partagé : ext3, ReiserFS (mauvaise idée), XFS, btrfs...

4.3.2 Autres systèmes de fichiers en réseaux

SMB (Server Message Block) est un protocole de Microsoft permettant de partager des ressources sur un réseau. Son implémentation Samba permet de partager des dossiers sur un réseau. Il peut être interface à Active Directory ou LDAP pour permettre un contrôle d'accès des clients sur les répertoires partagés.

Un NAS (Network-Attached Storage) est une machine spécifique, reliée au réseau, qui constitue elle-même le système de stockage des machines de ce réseau. À l'intérieur de cette machine on trouve souvent un système de stockage comme un RAID, permettant d'avoir de bonnes performances, un gros volume et un minimum de fiabilité. Certains NAS peuvent même être distribués sur plusieurs machines afin d'améliorer leurs performances et leur taille.

GlusterFS est un exemple de NAS. Chaque machine stockant des données est considérée comme une brique de stockage et gère localement ses données en utilisant le système de fichiers local. Le système GlusterFS a pour rôle de coordonner les données entre les briques de stockage (réplication, partage de charges...) et de permettre aux clients l'accès aux données stockées. Les clients accèdent aux données via le réseau, qui peut être TCP/IP ou un réseau rapide comme InfiniBand.

4.4 Techniques de stockage sûr

Les disques durs sont les points faibles d'un système informatique : c'est l'élément matériel qui a le plus petit temps moyen avant défaillance, c'est-à-dire que statistiquement, il y a une très grande probabilité pour que ce soit l'élément qui tombera en panne le premier. Pour palier à cette faiblesse, on met en place des stratégies de stockage sûr, qui permettent de ne pas perdre les données malgré les pannes qui peuvent survenir.

On modélise souvent la probabilité de panne des disques durs par une courbe dite "en baignoire", ou bathtub curve. Cette courbe correspond aux trois phrases de la vie d'un disque dur :

- lorsqu'un disque est neuf (moins de 3 mois environ), il a une forte probabilité de tomber en panne pour beaucoup de raisons : rodage, défauts de conception...
- puis il entre dans sa phase de croisière, pendant laquelle il peut tomber en panne, mais il n'a pas de raison particulière de le faire ;
- enfin, lorsqu'il est vieux (généralement au-delà de trois ans), il a à nouveau une forte probabilité de tomber en panne car il est vieux et usé.

Cette distribution de probabilité est représentée par la figure 2.

L'avantage de mutualiser les ressources de stockage en utilisant un système de stockage en réseau est alors que l'on peut mettre en place une solution de sauvegarde des données sur les disques en réseau, au lieu de la mettre en place sur les disques de chaque machine.

4.4.1 Réplication

La solution la plus basique consiste à effectuer une *réplication* pure et simple du disque. On a alors un deuxième disque, a priori identique ou au moins aussi gros, sur lequel on *recopie le contenu* du disque à sauvegarder.

La figure 3 représente la réplication d'un disque sur un autre disque. On a alors une copie *exacte* du disque A sur le disque B, à l'instant auquel la copie a été effectuée.

Différentes solutions sont possibles pour effectuer cette copie. L'outil Unix `dd` permet de faire une copie brute d'un fichier dans un autre fichier. Étant donné le fait que sous Unix, tout est fichier, y

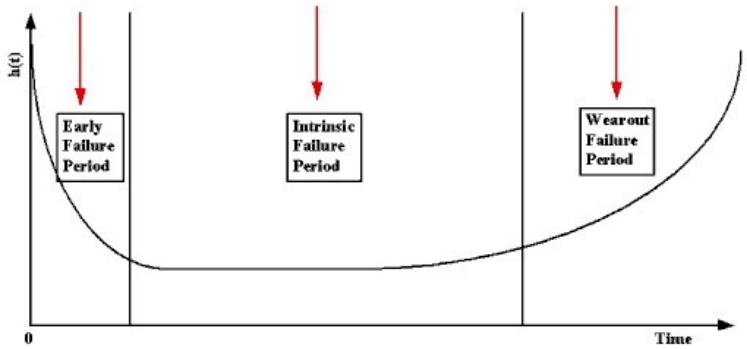


FIGURE 2 – La courbe en baignoire, ou bathtub curve, représentant la probabilité de panne d'un disque dur en fonction de son âge

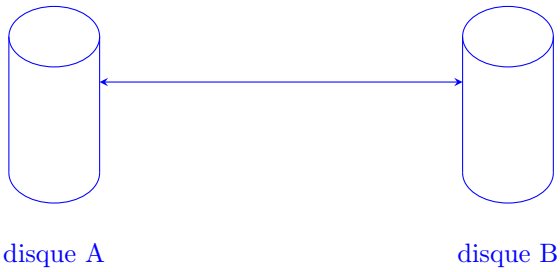


FIGURE 3 – Réplication d'un disque A sur un disque B

compris les disques durs (voir section 4.2.2), il peut prendre le contenu d'un disque pour le copier sur un autre disque ou dans un fichier.

Par exemple, la commande suivante réplique le contenu du disque `/dev/sda` sur le disque `/dev/sdb` :

```
1 root@maximum:~# dd if=/dev/sda of=/dev/sdb bs=1024
```

Le fichier d'entrée est spécifié par l'option `if` (input file), tandis que le fichier de sortie est spécifié par l'option `of` (output file). Attention, les disques utilisés ne doivent pas contenir de partitions montées sur le système au moment où la commande est exécutée.

La copie se fait bloc par bloc. L'option `bs` permet de spécifier la taille des blocs utilisés : il est conseillé d'utiliser une taille correspondant à celle utilisée par le système de fichiers copié.

On peut également écrire la copie dans un fichier :

```
root@maximum:~# dd if=/dev/sda of=svg_date.img bs=1024
```

Il est possible de compresser l'image obtenue en combinant avec un tube (pipe) la sortie de `dd` directement vers l'entrée de `gzip` (utilitaire de compression) :

```
root@maximum:~# dd if=/dev/sda bs=1024 | gzip > svg_date.img.gz
```

L'utilitaire `dd` permet de copier l'intégralité d'une partition ou d'un disque. Une autre possibilité est de rassembler des fichiers dans une archive avec l'utilitaire `tar`. On peut conserver les propriétés des fichiers avec les options `-posix -numeric-owner` :

```
coti@maximum:~$ tar -posix --numeric-owner czf /opt > /tempo/svg.tgz
```

L'opération de restauration est exactement l'opération inverse : on copie la sauvegarde sur le disque sur lequel on veut restaurer l'état. Le disque sauvegardé est recopié *intégralement*, y compris sa table des partitions.

Cette opération est représentée par la figure 4. Considérons que le disque A est tombé en panne. Son état demeure sauvegardé sur le disque B, que l'on utilise pour restaurer cet état sur le disque C. On se retrouve alors avec à nouveau deux disques contenant les données.

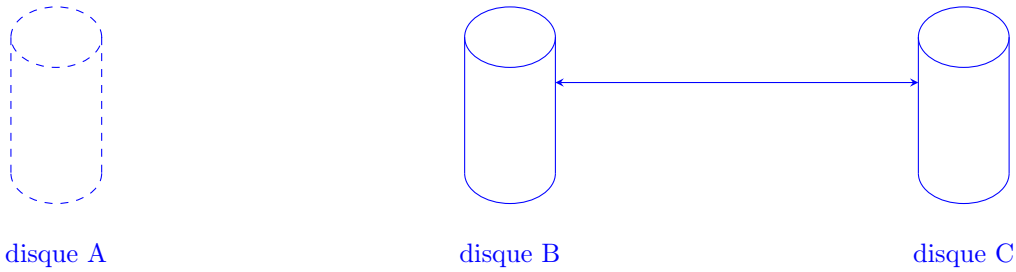


FIGURE 4 – Restauration sur le disque C de l'état du disque A sauvegardé sur le disque B

L'avantage de cette méthode est qu'elle permet d'avoir une sauvegarde brute de l'intégralité des données. Cependant, elle présente plusieurs inconvénients.

Elle nécessite notamment de doubler l'espace disque utilisés : pour un disque sauvegardé, il faut utiliser un espace correspondant exactement à sa capacité pour le sauvegarder. Ainsi, si l'on souhaite sauvegarder par exemple 3 disques de 1 To chacun, il est nécessaire de disposer de 3 To d'espace de sauvegarde, répartis par disques de 1 To chacun au minimum : soit 3 disques de 1 To chacun, soit 1 disque de 3 To, soit 1 disque de 2 To et 1 disque de 1 To.

Un autre inconvénient de cette approche est que l'on ne dispose pas d'une copie de sauvegarde à jour en temps réel, mais uniquement à jour au moment où la sauvegarde a été faite : toutes les modifications effectuées après le moment de la sauvegarde ne sont pas prises en compte dans cette dernière.

4.4.2 Sauvegarde incrémentale

La sauvegarde incrémentale consiste à sauvegarder uniquement les modifications qui ont eu lieu depuis la dernière sauvegarde. On commence par faire une sauvegarde complète du système, puis on ne sauvegarde que les modifications.

De temps en temps il est nécessaire de regarde une sauvegarde complète afin de pouvoir supprimer toutes les modifications : par exemple, lorsque la taille totale des modifications enregistrée dépasse la taille d'une sauvegarde complète.

On parle alors de *niveau de sauvegarde* : on sauvegarde les modifications qui ont été effectuées depuis N sauvegardes. Le niveau 0 correspond à une sauvegarde complète, non incrémentale. Le niveau 1 correspond à une sauvegarde des modifications ayant eu lieu depuis la dernière sauvegarde de niveau 0. Le niveau 2 correspond à une sauvegarde des modifications ayant eu lieu depuis la dernière sauvegarde de niveau 1, et ainsi de suite.

Par exemple, si nous avons un système composé des données suivantes :

```
AABBCCDDEEFFGGHHIIJJKK
```

On sauvegarde l'intégralité du système. Ensuite, on effectue une modification sur les données du système :

```
AABBCCDDEEOOGGHHIIJJKK
```

On effectue une sauvegarde incrémentale de niveau 1 : on n'enregistre que les modifications, c'est à dire, notons de la façon suivante "à 10 caractères du début, on enlève FF et on met OO à la place" :

```
+10 : -FF+OO
```

On effectue une nouvelle modification sur les données du système :

```
AAIICCDDEEOOGGHUUIJJKK
```

Et on effectue une nouvelle sauvegarde incrémentale :

```
+2 : -BB+II
+15 : -HI+UU
```

Considérons maintenant que le système de stockage vient d'être détruit et que que nous souhaitons retrouver les données. Nous repartons de la dernière sauvegarde intégrale et lui appliquons les modifications enregistrées dans les sauvegardes incrémentales :

```
AABBCCDDEEFFGGHHIIJJKK
  +10 : -FF+OO
AABBCCDDEEOOGGHHIIJJKK
  +2 : -BB+II
  +15 : -HI+UU
AAIICCDDEEOOGGHUUIJJKK
```

On retrouve bien les données telles qu'elles étaient lorsque le système a été détruit.

Dans le cas de la deuxième sauvegarde incrémentale, si l'on avait effectué une sauvegarde de niveau 2, celle-ci aurait été composée des modifications effectuées depuis l'avant-dernière sauvegarde :

```
  +10 : -FF+OO
  +2 : -BB+II
  +15 : -HI+UU
```

L'avantage est important en termes de gain de place et de rapidité des opérations de sauvegarde. Cependant, l'image du système a besoin d'être reconstruite en repartant de la dernière sauvegarde complète et en lui appliquant toutes les modifications ultérieures. On ne peut pas récupérer un seul fichier.

Les implémentations GNU de la commande `tar` permettent par exemple d'effectuer une sauvegarde incrémentale en se basant sur un fichier de status.

L'outil `rsnapshot` permet d'effectuer des sauvegardes incrémentales de systèmes de fichiers locaux ou distants (sur le réseau). Il peut être configuré pour ne sauvegarder que certains répertoires, et utiliser des niveaux de sauvegarde différents suivant la fréquence concernée (horaire, quotidien, hebdomadaire).

L'outil `dump` est spécifique aux systèmes de fichiers ext2/ext3/ext4. On utilise l'outil correspondant `restore` pour restaurer le système de fichiers sauvegardé.

On peut spécifier le niveau de sauvegarde à effectuer. L'exemple suivant sauvegarde le contenu du répertoire `/home` en effectuant une sauvegarde complète (première ligne), une sauvegarde de niveau 1, une sauvegarde de niveau 2 et une sauvegarde de niveau 3.

```
root@abidjan:~# dump -0uf data0.bkp /home
root@abidjan:~# dump -1uf data1.bkp /home
root@abidjan:~# dump -2uf data2.bkp /home
root@abidjan:~# dump -3uf data3.bkp /home
```

On obtient alors des fichiers contenant les dernières modifications et des données d'indexation internes. C'est pourquoi les fichiers de sauvegarde incrémentale sont plus gros que le volume des modifications effectuées.

```
root@abidjan:~# ls -l
total 18322096
-rw-r--r-- 1 root root     2764800 avril 26 00:46 data1.bkp
-rw-r--r-- 1 root root     2764800 avril 26 00:50 data2.bkp
-rw-r--r-- 1 root root 18737960960 avril 26 00:45 data.bkp
```

L'outil `restore` est le pendant de `dump` permettant de restaurer les données sauvegardées. Il offre un mode interactif permettant de se déplacer parmi les fichiers sauvegardés (option `-i`). L'option `-t` permet d'examiner la sauvegarde en listant son contenu.

La récupération des données se fait de la même façon, avec les options `r`. On passe les sauvegardes les unes après les autres, dans le même ordre que celui dans lequel elles ont été faites : d'abord la sauvegarde initiale, puis les sauvegardes incrémentales ultérieures. Il est à noter qu'un fichier `restoresymtable` est créé par `restore` : dedans sont stockées des informations entre les restaurations successives.

4.4.3 Redondance : le RAID

Afin de palier aux inconvénients de la réplication, on peut utiliser une approche plus fine de la redondance. C'est par exemple ce qui est fait avec le stockage *RAID* (Redundant Array of Independant Disks). À l'origine, l'acronyme signifiait Redundant Array of Inexpensive Disks : il s'agissait de construire un système de stockage rendu fiable et performant en utilisant un ensemble de disques durs bon marchés, et donc lents et non fiables. Comme aujourd'hui tous les disques durs, bon marchés ou non, sont considérés comme non fiables et lents par rapport au reste des composants, il n'est plus question de disques "inexpensive".

Il existe plusieurs types de RAID, désignés par des numéros : RAID 0, RAID 1... à RAID 6. On peut également les combiner : RAID 1 de RAID 5 appelé RAID 0+5, par exemple.

Les principes de base du RAID sont :

— Il est possible de lire sur plusieurs disques à la fois, ou indépendemment les uns des autres
— Il est possible d'écrire sur plusieurs disques à la fois, ou indépendemment les uns des autres
— Lorsque l'on lit ou que l'on écrit sur plusieurs disques à la fois, on peut écrire autant de fois la même information, ou découper l'information et la lire ou l'écrire en parallèle sur ces disques
— Il est possible d'écrire une information plusieurs fois, ou de récupérer une information en en combinant plusieurs

Les plus courants sont les RAID 0, 1 et 5. Les RAID 2 à 4 sont obsolètes, le RAID 6 est peu utilisé car complexe et, par conséquent, trop lent.

RAID 0 Le RAID 0 consiste à écrire puis à lire en parallèle sur plusieurs disques durs. Il ne fiabilise pas le système de stockage, mais le rend *plus rapide*. Ainsi, dans un monde parfait, le temps nécessaire à la lecture ou à l'écriture d'un volume de données sur D disques serait divisé par D.

Par exemple, imaginons que nous devions écrire la suite de données `ABCDEFGHI` sur trois disques en RAID 0. Le flux de données à écrire est réparti entre les trois disques : `A` va sur le disque 1, `B` va sur le disque 2, `C` va sur le disque 3, `D` va sur le disque 1, `E` va sur le disque 2, et ainsi de suite. À la fin de l'écriture, le disque 1 contient `ADG`, le disque 2 contient `BEH` et le disque 3 contient `CFI`. Il n'aura été nécessaire d'écrire que 3 lettres sur chacun des disques pour écrire au total 9 lettres. Cette situation est représentée par la figure 5.

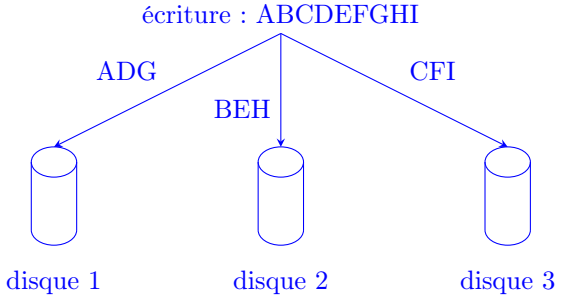


FIGURE 5 – RAID 0 : écriture en parallèle sur les disques 1, 2 et 3.

Ainsi, si l'on veut écrire un volume de données V sur D disques durs, chaque disque dur recevra le volume V/D. Le temps nécessaire à son écriture, si l'on écrit à la vitesse de τ octets par seconde, sera

alors idéalement de $V/(D * \tau)$ au lieu de V/τ.

La lecture se fait, de même, en parallèle sur tous les disques en même temps. Le temps de lecture est également idéalement divisé par le nombre de disques utilisés. La lecture est représentée par la figure 6.

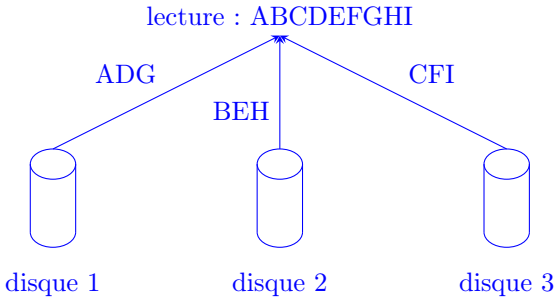


FIGURE 6 – RAID 0 : lecture en parallèle sur les disques 1, 2 et 3.

Le RAID 0 ne rend pas le système de stockage plus sûr : en effet, il n'apporte pas de redondance. Si un disque tombe en panne, les données du système ne sont pas récupérables. Il le rend uniquement plus rapide en lecture et en écriture.

L'espace de stockage du système est limité par la taille du plus petit disque. Lorsqu'un des disques est plein, il n'est plus possible d'écrire dans le système de stockage. Si on a D disques de capacités C_i, i désignant le disque, l'espace total de stockage disponible est donc de $D * min(C_i)$.

Par exemple, si un système est composé de deux disques de 3 To et d'un disque de 2 To, le disque limitant sera celui de 2 To : le volume total disponible dans ce système en RAID 0 est alors de 6 To.

RAID 1 Le RAID 1 est une réplication pure et simple. Lorsque l'on écrit une information sur un disque, on l'écrit également, en même temps, sur un ou plusieurs autre(s) disque(s). On a alors, à tout moment, exactement la même information sur tous les disques : on parle alors de *miroir*.

Ainsi, si l'on écrit les données `ABCD` sur un système de stockage composé de deux disques en RAID 1, les deux disques recevront `ABCD`. Une écriture sur un système en RAID 1 composé de deux disques est représentée par le schéma de la figure 7.

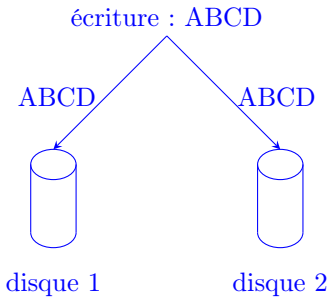


FIGURE 7 – RAID 1 : écriture simultanée sur le disque 1 et le disque 2.

Les deux disques disposent alors de la même information. La lecture peut donc s'effectuer sur l'un ou sur l'autre. Généralement, les systèmes RAID 1 choisissent de lire sur le disque qui répond le plus rapidement. Cela présente notamment l'avantage d'équilibrer la charge (un disque occupé répondra plus lentement donc on ira lire sur un autre) et de ne pas charger un disque présentant des faiblesses. Il est également possible de tirer au sort aléatoirement le disque : ainsi, statistiquement, les lectures sont réparties équitablement entre les disques.

Le RAID 1 sert uniquement à fiabiliser le système de stockage : il n'accélère pas les lectures ni les écritures. Dans un monde parfait, la lecture et l'écriture se font à la même vitesse que sur un seul

disque. Il est cependant très efficace pour rendre résilient le système de stockage : si l'on dispose de D disques, on peut tolérer jusqu'à $D-1$ pannes de disques sans pertes de données.

Son espace de stockage est, comme pour le RAID 0, limité par la taille du plus petit disque. Cependant, contrairement au RAID 0, le RAID 1 redonde les données sur tous les disques : l'espace disponible pour le stockage est donc de $min(C_i)$.

RAID 5 Le RAID 5 a pour but de combiner le RAID 0 et le RAID 1 : il permet la lecture et l'écriture en parallèle sur plusieurs disques, tout en apportant de la redondance pour permettre de tolérer les pannes de disques.

Il utilise pour cela un *système de détection et correction d'erreur par parité*. Un certain nombre de disques reçoivent non pas des informations à écrire, mais un `bloc de parité` permettant de reconstituer celles-ci en cas de panne d'un ou plusieurs disques. Considérons un système de stockage constitué de D disques. Le flux de données à écrire est découpé en $(D-n)$ parties écrites en parallèle sur $(D-n)$ disques. Puis un calcul de parité est effectué sur ces $(D-n)$ portions de données et le résultat est écrit sur les n disques restants.

Par exemple, si l'on dispose d'un système composé de 4 disques sur lequel on souhaite écrite `ABCDEFGHI` en utilisant un bloc de parité : l'écriture des données se fera alors sur 3 disques. Les 3 premières lettres sont écrites sur les 3 premiers disques. Puis l'octet de parité $A \oplus B \oplus C$ est calculé et écrite sur le dernier disque.

À des fins d'équilibrage de charge, l'octet de parité n'est pas toujours écrit sur le même disque : celui-ci est sélectionné de façon rotationnelle, à la manière d'un round-robin. Ainsi, pour la suite de notre exemple, les 3 octets suivants `DEF` sont donc écrits respectivement sur les disques 2 à 4, et l'octet de parité $D \oplus E \oplus F$ est écrit sur le disque 1. De même, l'octet de parité $G \oplus H \oplus I$ est écrit sur le disque 2. Cet exemple est illustré par la figure 8.

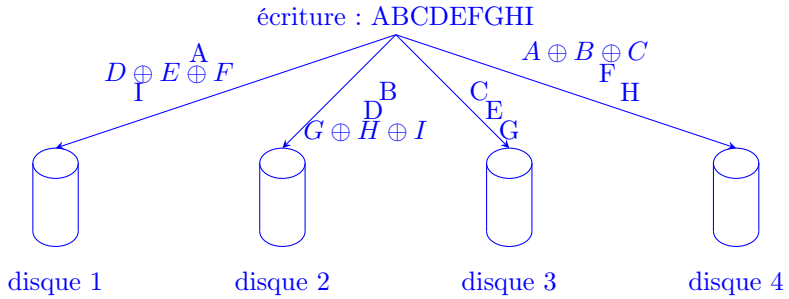


FIGURE 8 – RAID 5 : Écriture sur quatre disques avec un bloc de parité.

La lecture se fait en parallèle sur les disques en supprimant les blocs de parité. La lecture pour l'exemple précédent est illustrée par la figure 9.

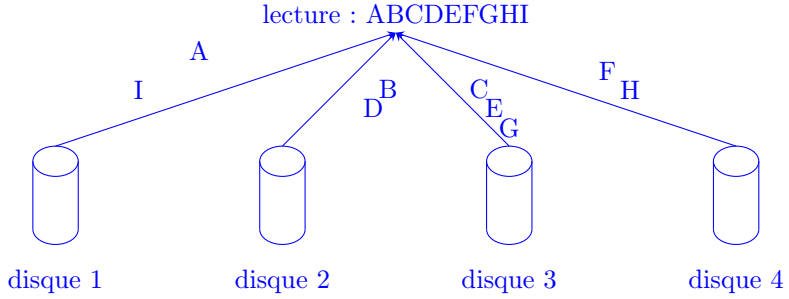


FIGURE 9 – RAID 5 : Lecture sur quatre disques avec un bloc de parité.

En cas de panne d'un disque dur, les données qui s'y trouvaient sont reconstituées en utilisant les données écrites en même temps sur les autres disques et le bloc de parité. Si on a des données A_i et un bloc de parité X et que l'on a perdu un bloc de donnée A_ns, on reconstitue le bloc perdu en le remplaçant par le bloc de parité dans le calcul de parité : $A_n = A_0 \oplus ... \oplus A_x ... \oplus A_{(D-n)}$

Dans notre exemple, si le disque 2 tombe en panne, on reconstitue ses données : en appelant P les blocs de parité, on a alors :

— $A \oplus P_{(ABC)} \oplus C = B$
— $P_{(DEF)} \oplus E \oplus F = D$
— $G \oplus H \oplus I = P_{(GHI)}$

Le RAID 5 cumule dont les avantages du RAID 0 et du RAID 1 : on lit et on écrit en parallèle sur plusieurs disques, et le système est capable de tolérer des pannes de disques.

Quantitativement, si l'on considère un système de D disques utilisant n blocs de parité, celui-ci est capable de tolérer la panne de n disques. Dans un monde parfait, la lecture et l'écriture sont accélérées par un facture $(D-n)$: si l'on veut écrire un volume de données V sur ce système, chaque disque dur recevra le volume $V/(D-n)$. Le temps nécessaire à son écriture, si l'on écrit à la vitesse de τ octets par seconde, sera alors idéalement de $V/((D-n)*\tau)$ au lieu de V/τ.

Ici encore, l'espace nécessaire est limité par le plus petit disque disponible. L'espace disponible dans le système est donc ici de $(D-n)*min(C_i)$.

Grâce à la formule de reconstitution des données, on voit que le système peut tolérer autant de pannes de disques qu'il y a de blocs de parité. Ainsi, un système de D disques avec n blocs de parité peut tolérer au maximum la défaillance simultanée d'au plus n disques sans perte de données

RAID 01 Le RAID 01 est une combinaison du RAID 0 et du RAID 1. L'idée est d'écrire en miroir sur plusieurs systèmes qui, eux, écrivent en parallèle sur plusieurs disques. L'écriture sur un exemple de système RAID 01 est représenté par la figure 10. La lecture se fait en parallèle sur deux disques. Ce système tolère autant de défaillances qu'il y a de systèmes en miroir dans la partie en RAID 0.

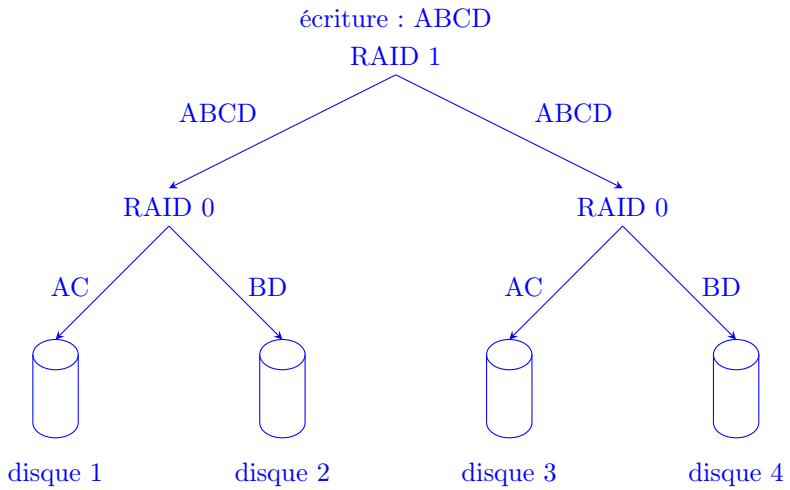


FIGURE 10 – RAID 01 : écriture sur quatre disques, en écrivant en miroir sur deux systèmes de deux disques en parallèle.

RAID 10 Le RAID 10 est l'inverse du RAID 01 : on écrit en parallèle sur plusieurs systèmes qui sont eux composés de plusieurs disques en miroir. Un exemple d'un tel système est représenté par la figure 11. L'avantage, en terme de flexibilité, est que chaque système en RAID 1 dispose de la fiabilité qu'il veut/peut (en terme de redondance) sans déséquilibrer le reste du système.

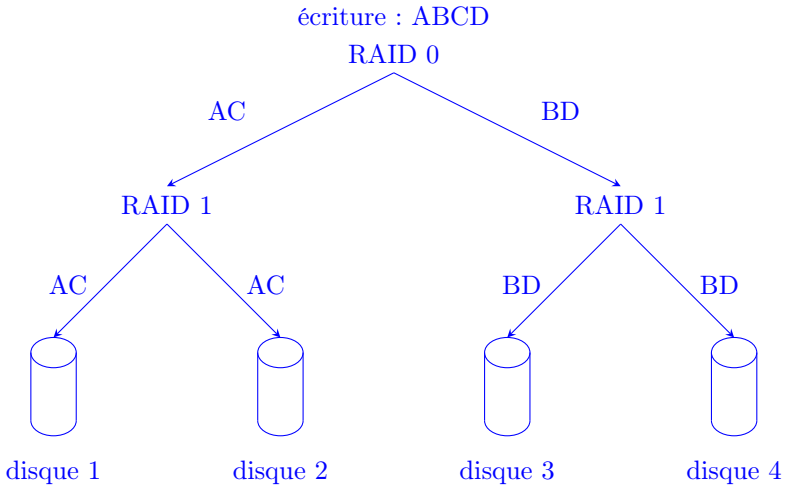


FIGURE 11 – RAID 10 : écriture sur quatre disques, en écrivant en parallèle sur deux systèmes composés chacun de deux disques en miroir.

Autres niveaux de RAID D'autres niveaux de RAID plus exotiques et d'autres combinaisons existent, comme le RAID 7, le RAID TP (Triple Parité), les RAID 15, 50 ou autres. Il est possible d'innover et de mettre en place des solutions ad hoc présentant le meilleur compromis pour une situation donnée entre le coût, la résilience et les performance.

5 Métrologie et alertes : Mesurer l'état d'une machine

En étant simplement sur une machine, il existe une grande quantité d'outils locaux pour surveiller son état. Il peut s'agir de sondes matérielles ou logicielles.

Les sondes logicielles sont généralement fournies par le système d'exploitation, afin de donner une idée de ce qui est en train de tourner et de l'état de l'utilisation des ressources. C'est notamment l'utilité du pseudo système de fichiers `/proc`. Les sondes matérielles nécessitent du matériel adapté, comme par exemple un capteur de température.

5.1 Charge d'une machine

La charge CPU d'une machine peut être visualisée par la commande `top`. On peut y voir, parmi d'autres informations, la liste des processus exécutés sur la machine et, pour chaque processus, quelques informations : pourcentage CPU utilisé, mémoire utilisée, temps écoulé depuis le début de l'exécution... On dispose également d'informations globales sur la machine, comme la charge moyenne sur les 1, 5, et 15 dernières minutes. L'affichage peut être trié selon un critère particulier.

```
top - 14:47:39 up 14 days,  2:35, 10 users,  load average: 0,96, 0,99, 0,92
Tasks: 244 total,   2 running, 242 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,3 us,  0,5 sy,  0,0 ni, 96,8 id,  0,5 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   8153328 total,  7021852 used,  1131476 free,   196668 buffers
KiB Swap:  7811068 total,     2644 used,  7808424 free,  2781416 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
28710 coti      20   0  986m 148m  23m S   7,0  1,9   0:36.82 chrome
 3529 coti      20   0 1189m  68m  11m S   3,7  0,9  35:04.63 chrome
 2989 coti       9 -11  434m 8824 5584 S   2,3  0,1  22:57.37 pulseaudio
 3167 coti      20   0 1308m 352m  48m S   2,3  4,4 134:44.69 chrome
 1734 root      20   0  260m 153m  27m S   1,7  1,9 185:30.21 Xorg
27381 coti      20   0  993m 127m  28m S   1,3  1,6   1:26.38 chrome
29027 coti      20   0  966m  95m  22m S   1,0  1,2   0:04.14 chrome
22163 coti      20   0  231m  46m  14m S   0,7  0,6   0:14.43 emacs
27744 coti      20   0  945m 124m  20m S   0,7  1,6   0:01.98 chrome
 1788 avahi     20   0 35928 3700 1488 S   0,3  0,0  23:25.46 avahi-daemon
 4301 coti      20   0  386m  24m  12m S   0,3  0,3   0:55.60 gnome-terminal
```

5.2 État de la mémoire vive

On peut voir la quantité de mémoire utilisée avec la commande `free`. Le résultat donne des informations sur la mémoire vive physique et sur l'espace d'échange (partition swap, utilisée lorsque la mémoire physique est pleine). Les quantités sont données en ko par défaut, sinon octets, Mo, Go ou compatible avec les humains, respectivement avec les options `-b`, `-m`, `-g` ou `-h`.

```
coti@maximum:~$ free -h
             total       used       free     shared    buffers     cached
Mem:          7,8G       7,0G       827M          0B       192M       2,7G
-/+ buffers/cache:       4,1G       3,7G
Swap:         7,4G       2,6M       7,4G
```

Dans l'exemple ci-dessus, on voit que la machine dispose au total de 7,8 Go, dont 7 sont utilisés. La partition d'échange est de 7,4 Go, dont une quantité négligeable est utilisée (1,6 Mo).

Il est possible de voir l'utilisation qui est faite de la mémoire, et son évolution au cours du temps, avec la commande `vmstat`.

```
coti@maximum:~$ vmstat 2 5
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  1   2644 802256 196804 2802480    0    0     0     1    4    2  1  0 99  0
 0  0   2644 789608 196804 2803268    0    0     0     0 6362 9583  5  2 91  2
 0  0   2644 790072 196804 2803332    0    0     0     0 3893 6205  3  1 94  2
 0  1   2644 774516 196804 2803684    0    0     0     0 8501 12415  6  2 87  5
 1  0   2644 770200 196804 2803800    0    0     0     0 3082 5239  3  1 85 11
```

5.3 Communications

Il est également possible de surveiller l'état des communications effectuées par le système. Plusieurs outils sont disponibles selon le type d'informations recherchées et le type de communications que l'on veut surveiller.

5.3.1 Communications inter-processus

Les communications inter-processus, ou IPC V5 (Inter Process Communications), peuvent être listées avec la commande `ipcs`.

```
coti@maximum:~$ ipcs
------ Segment de mémoire partagée --------
clé        shmid      propriétaire perms      octets     nattch     états
0x00000000 0          coti       600        393216     2          dest
0x00000000 32769      coti       600        393216     2          dest
0x00000000 342130690  coti       777        2050328    2          dest
0x00000000 273055747  coti       777        561600     2          dest
[...]
------ Tableaux de sémaphores --------
clé        semid      propriétaire perms      nsems
------ Queues de messages --------
clé        msqid      propriétaire perms      octets utilisés messages
```

On voit ici quelle est la taille de l'IPC, son identifiant noyau, son propriétaire et d'autres informations sur chaque élément de communication inter-processus. Les diverses options permettent d'obtenir plus d'informations sur un type d'IPC en particulier : par exemple, l'option `-S` permet d'obtenir des informations détaillées sur les sémaphores utilisés dans le système, tandis que l'option `-M` permet d'obtenir des informations détaillées sur les segments de mémoire partagée.

5.3.2 Fichiers ouverts

On peut obtenir des informations sur les fichiers ouverts dans le système en utilisant la commande `lsof`. Étant donné le fait que sous Unix tout est fichier, cette commande donne par la même occasion des informations notamment sur les sockets réseau ouvertes (option `-i`).

```
coti@maximum:~$ lsof -i
COMMAND     PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
icedove-b  3160 coti   56u  IPv4 1816971      0t0  TCP maximum:45516->wg-in-f109.1e100.net:imaps (ESTABLISHED)
icedove-b  3160 coti   58u  IPv4   18438      0t0  TCP maximum:55337->mail.iutv.univ-paris13.fr:imaps (ESTABLISHED)
icedove-b  3160 coti   59u  IPv4 2022424      0t0  TCP maximum:56597->mail:imaps (ESTABLISHED)
icedove-b  3160 coti   69u  IPv4 2022443      0t0  TCP maximum:32872->mail.iutv.univ-paris13.fr:imaps (ESTABLISHED)
icedove-b  3160 coti   76u  IPv4 2496423      0t0  TCP maximum:32953->par08s09-in-f19.1e100.net:http (ESTABLISHED)
icedove-b  3160 coti   79u  IPv4 1162259      0t0  TCP maximum:59881->mail:imaps (ESTABLISHED)
```

Dans l'exemple ci-dessus, on voit notamment le pid du processus qui a ouvert le fichier : c'est une information extrêmement intéressante pour savoir quel processus est en train de lire ou d'écrire dans un fichier.

En lui passant en paramètre un chemin vers un répertoire, on obtient des informations sur les fichiers ouverts se trouvant dans ce répertoire et ses sous-répertoires.

```
coti@maximum:~$ lsof /

COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
bash       1746 coti  rtd    DIR    8,1     4096       2 /
bash       1746 coti  txt    REG    8,1   975488 5373975 /bin/bash
bash       1746 coti  mem    REG    8,1    47616 4769217 /lib/x86_64-linux-gnu/libnss_files-2.13.so
bash       1746 coti  mem    REG    8,1    43552 4769213 /lib/x86_64-linux-gnu/libnss_nis-2.13.so
bash       1746 coti  mem    REG    8,1    89056 4769211 /lib/x86_64-linux-gnu/libnsl-2.13.so
```

5.3.3 Communications réseaux

Une commande très utilisée par les administrateurs systèmes et les administrateurs réseaux est `netstat` : elle permet d'obtenir des informations sur les sockets ouvertes. Cette commande dispose d'un certain nombre d'options très utiles, parmi lesquelles :

- Possibilité de préciser le protocole : `-u` pour UDP, `-t` pour TCP
- Donne le PID et le nom du programme utilisant la socket
- Port local, adresse et port distants
- État de la socket

```
coti@maximum:~$ netstat -lapute
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat        User       Inode      PID/Program name
tcp        0      0 *:sunrpc                *:*                     LISTEN      root       3906       -
tcp        0      0 *:ssh                   *:*                     LISTEN      root       7730       -
tcp        0      0 localhost:ipp           *:*                     LISTEN      root       1861089    -
tcp        0      0 localhost:smtp          *:*                     LISTEN      root       6142       -
tcp        0      0 *:841                   *:*                     LISTEN      root       10374      -
tcp        0      0 *:33388                 *:*                     LISTEN      root       10711      -
tcp        0      0 *:42989                 *:*                     LISTEN      statd      3925       -
tcp        0      0 maximum:41813           wg-in-f125.1e100.n:5223 ESTABLISHED coti       1874088    27428/telepathy-gab
tcp        0      0 maximum:54638           magi.univ-paris13.:2822 ESTABLISHED coti       2025047    31633/ssh
tcp        0      0 maximum:36752           we-in-f109.1e100.:imaps ESTABLISHED coti       1872420    3160/icedove-bin
```

On voit ici les sockets TCP ouvertes qui écoutent les connexions entrantes : ce sont les sockets à l'état LISTEN. Les sockets à l'état ESTABLISHED correspondent à des communications TCP établies entre deux machines. On voit notamment deux connexions entre maximum et une machine du domaine 1e100.net (qui appartient à Google), et une connexion avec magi.univ-paris13.fr. Lorsque le port correspond à un port standard connu par le système, il est remplacé par son nom : c'est le cas notamment du port ssh ou sunrpc.

5.4 État des cartes réseaux

On peut voir l'état des interfaces réseaux du système avec la commande `ifconfig`. Cette commande peut être utilisée par le super-utilisateur pour configurer les cartes réseaux. Elle demeure utilisable en lecture seule par les autres utilisateurs pour obtenir des informations sur l'état des cartes réseaux. À noter qu'elle n'est généralement pas dans le PATH des utilisateurs normaux : il est nécessaire de taper le chemin entier.

```
coti@maximum:~$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr d4:be:d9:9c:7a:9c
          inet adr:10.10.0.217  Bcast:10.10.255.255  Masque:255.255.0.0
          adr inet6: fe80::d6be:d9ff:fe9c:7a9c/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:42594748 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28853636 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:18842740245 (17.5 GiB)  TX bytes:26078786297 (24.2 GiB)
          Interruption:20 Mémoire:e4c00000-e4c20000
```

Dans l'exemple ci-dessus, on voit que la machine ne dispose que d'une seule interface réseau : eth0. Outre les informations classiques (type de réseau, adresse matérielle, adresse protocolaire...), on voit le nombre de paquets reçus et envoyés, ainsi que le nombre d'erreurs.

5.5 Surveillance des disques

Il existe plusieurs outils permettant de surveiller l'état des disques et leur utilisation. Le recoupement des informations fournies par ces outils constitue également une source d'information.

5.5.1 Espace restant

On peut obtenir l'espace restant sur les disques grâce à la commande `df` (disk free). Le résultat est donné par défaut en multiple du nombre de blocs du système de fichiers.

```
coti@maximum:~$ df
Sys. fich.                     1K-blocks      Util. Disponible Uti% Monté sur
/dev/sda1                       96120588   12105232   79132620  14% /
tmpfs                            1630668      92104    1538564   6% /tmp
/dev/sdb5                      384499764     203560  364764684   1% /home
lipn-sfa:/export4/vol04/coti  1922471424 1629569536  195246080  90% /users/coti
```

Dans l'exemple ci-dessus, on voit deux partitions `/dev/sda1` et `/dev/sdb5`, montés dans l'arborescence du système de fichier respectivement aux points de montage `/` et `/home`. Ils sont utilisés respectivement à 14% et 1%.

5.5.2 Espace utilisé

On peut obtenir l'espace utilisé par un fichier avec la commande `du` (disk used). En lui passant un répertoire, on peut obtenir l'espace utilisé par ce répertoire et son contenu, en comprenant ses sous-répertoires et toute l'arborescence qui en découle.

```
coti@maximum:~$ sudo du -sh /
```

La commande ci-dessus permet d'obtenir l'espace utilisé par toute l'arborescence de fichiers. On peut ainsi obtenir l'espace utilisé par l'ensemble de fichiers se trouvant sur un disque dur. Attention alors : il faut se méfier si les résultats donnés par `du` et `df` ne sont pas cohérents. Il peut s'agir d'un programme qui plante au milieu d'opérations disques, ou d'un piratage qui dépose des fichiers qui ne sont pas visibles car les outils sont corrompus (piratage de type rootkit).

5.5.3 Entrées-sorties

On peut obtenir des statistiques d'entrée-sorties sur les disques en utilisant la commande `iostat`. Cela permet notamment de voir si un disque a une activité anormalement élevée (programme buggé qui écrit sans arrêt), ou si il existe un déséquilibre entre plusieurs disques entre lesquels la charge est censée être répartie. Cela peut également être un outil de diagnostic en cas de lenteur.

```
coti@maximum:~$ iostat
Linux 3.2.0-3-amd64 (maximum)         27/09/2012         _x86_64_        (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,71    0,00    0,14    0,22    0,00   98,93

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0,33         1,19         5,45    1463193    6703576
sdb               0,00         0,01         0,00       6456        120
```

5.6 Le pseudo-système de fichiers /proc

`/proc` est un *pseudo-système de fichiers* : on y accède (en lecture seule) comme à un système de fichiers monté comme un système de fichiers, mais ça n'en est pas un.

Il est utilisé par le noyau pour représenter des informations sur les processus qui tournent. Les informations relatives à chaque processus de pid `<pid>` sont situées dans un répertoire `/proc/<pid>`.

On y trouve entre autres :

— `cmdline` : ligne de commande ayant lancé le processus
— `statm` : statistiques sur la mémoire (taille, pages partagées...)
— `fd` : sous-répertoire contenant des liens vers tous les fichiers ouverts par le programme

```
coti@maximum:~$ sudo ls /proc/772
attr        coredump_filter  io          mountstats     pagemap      stat
autogroup   cpuset           limits      net            personality  statm
auxv        cwd              loginuid    ns             root         status
cgroup      environ          maps        numa_maps      sched        syscall
clear_refs  exe              mem         oom_adj        sessionid    task
cmdline     fd               mountinfo   oom_score      smaps        wchan
comm        fdinfo           mounts      oom_score_adj  stack
```

5.7 Sondes de températures

Les constructeurs placent souvent des sondes de températures à des endroits critiques : CPU, disques, GPU... Sous Linux, ces sondes peuvent être lues en utilisant l'utilitaire `lm_sensors`.

Il existe généralement deux limites : la limite haute et la limite critique. La limite haute est une sorte d'avertissement : quand elle est atteinte, le système met en place des mécanismes visant à la faire diminuer (par exemple une baisse de la fréquence CPU, pour les processeurs disposant de cette fonctionnalité). La limite critique ne devrait jamais être atteinte. Lorsqu'elle l'est, c'est une vraie alerte. Beaucoup de systèmes s'éteignent lorsqu'ils l'atteignent.

```
coti@maximum:~$ sensors
coretemp-isa-0000
Adapter: ISA adapter
Physical id 0:  +34.0 C  (high = +80.0 C, crit = +98.0 C)
Core 0:         +30.0 C  (high = +80.0 C, crit = +98.0 C)
Core 1:         +31.0 C  (high = +80.0 C, crit = +98.0 C)
Core 2:         +32.0 C  (high = +80.0 C, crit = +98.0 C)
Core 3:         +32.0 C  (high = +80.0 C, crit = +98.0 C)
```

6 Effectuer des tâches répétitives à des instants prédéfinis

Il peut être utile de planifier des tâches qui vont être répétées à des moments précis : par exemple, une sauvegarde des fichiers toutes les heures, une mise à jour du système toutes les semaines, une indexation des fichiers d'une partition tous les jours...

Sous Linux, l'outil `crontab` permet de planifier des tâches en spécifiant le moment auquel elles seront déclenchées. On peut notamment définir des tâches toutes les minutes, toutes les heures, tous les jours, toutes les semaines, tous les mois ou tous les ans.

Ces tâches sont définies de plusieurs façons :

— Dans un fichier `/etc/crontab` : ce sont les définitions des tâches répétitives du système ;

— Dans des répertoires `/etc/cron.monthly`, `/etc/cron.weekly`, `/etc/cron.daily` et `/etc/cron.hourly` : ce sont des tâches qui seront déclenchées, comme les noms des répertoires l'indiquent, tous les mois, toutes les semaines, tous les jours et toutes les heures. Elles doivent être déclenchées explicitement par `crontab`, mais on les range ici pour les rassembler ;

— Dans les crontab des utilisateurs.

Certains utilisateurs peuvent ne pas avoir le droit d'utiliser `crontab` : leur liste est donnée dans le fichier `/etc/cron.deny`. On peut aussi définir une liste d'utilisateurs autorisés explicitement à l'utiliser : on la donne dans le fichier `/etc/cron.allow`, et dans ce cas, tous les utilisateurs qui ne sont pas autorisés explicitement sont interdits.

La commande `crontab -l` permet de voir les tâches définies pour un utilisateur. Il est possible de les éditer avec `crontab -e`. Un éditeur de texte s'ouvre alors et offre la possibilité de définir les tâches.

Le fichier `/etc/crontab` et ce qui s'ouvre avec `crontab -e` ont la même syntaxe. On y trouve un tableau qui définit le moment où sont exécutées ces tâches et comportant six ou sept champs :

— Les cinq premiers champs définissent le moment où la tâche doit être exécutée ;

— Uniquement pour le fichier `/etc/crontab`, qui contient des définitions pour tout le système, on trouve l'identité d'utilisateur sous laquelle la tâche doit être exécutée ;

— Le dernier champ spécifie la tâche à exécuter.

Les cinq champs qui définissent le moment auquel la tâche doit être exécutée sont dans l'ordre suivant : minute, heure, jour du mois, mois, jour de la semaine :

— Le champ "minute" (`m`) donne la minute à laquelle la tâche doit être exécutée. Par exemple, si l'on souhaite qu'un tâche soit exécutée à la minute 5 (l'heure est à déterminer par les autres champs), cette colonne doit contenir la valeur 5. Par conséquent, cette valeur est comprise entre 0 et 59.

— le champ "heure"(`h`) donne l'heure à laquelle la tâche doit être exécutée. Par exemple, si l'on souhaite que la tâche soit exécutée à minuit, cette colonne doit contenir la valeur 5. Par conséquent, cette valeur est comprise entre 0 et 23. Attention, cela ne concerne que l'heure : si l'on souhaite que la tâche s'exécute à minuit mile, il faudra mettre 0 dans la colonne des minutes et 0 dans la colonne des heures.

— le champ "jour du mois" (`dom`) donne le jours dans le mois auquel la tâche doit s'exécuter. Par exemple, si l'on souhaite qu'une tâche s'exécute le premier du mois, cette colonne doit contenir la valeur 5. Par conséquent, cette valeur est comprise entre 1 et 31.

— Le champ "mois" (`mon`) donne le mois pendant lequel la tâche doit s'exécuter. On peut utiliser des chiffres ou les trois premières lettres du nom du mois (en anglais naturellement). Par exemple, si l'on souhaite qu'une tâche s'exécute au mois de février, cette colonne doit contenir la valeur 2 ou Feb. Par conséquent, cette valeur est comprise entre 1 et 12.

— Le champ “jour de la semaine” (`dow`) donne le jour de la semaine pendant lequel la tâche doit s’exécuter. On peut utiliser des chiffres ou les trois premières lettres du nom du jour (en anglais naturellement). Par exemple, si l’on souhaite qu’une tâche s’exécute le mercredi, cette colonne doit contenir la valeur 3 ou Wed. Par conséquent, cette valeur est comprise entre 0 et 7, 0 et 7 désignant tous les deux le dimanche.

On peut également utiliser la valeur particulière “`*`” (astérisque), qui désigne “toutes les valeurs”. Par exemple, “`*`” dans la colonne des heures signifie que la tâche doit s’exécuter toutes les heures.

Il est à noter qu’il est possible de définir des intervalle ou des listes pour ces valeurs. Par exemple, `1-5` dans la colonne du jour de la semaine signifie “du lundi au vendredi”, ou `1,4,7,10` dans la colonne du mois signifie “en janvier, avril, juillet et octobre”.

Par exemple, si l’on trouve dans le `crontab` d’un utilisateur le champ suivant, un affichage sera effectué tous les jours de la semaine à 7 :00 :

```
0 7 * * 1-5    echo "Debout feignasse !"
```

La définition suivant exécute une tâche toutes les minutes :

```
* * * * *    $HOME/script.sh
```

Dans le fichier `/etc/crontab`, on doit en plus définir l’identité de l’utilisateur sous laquelle la tâche doit être exécutée. Par exemple, pour exécuter `updatedb` (qui met à jour une base de donnée d’indexation des fichiers) tous les jours à minuit sous l’identité du super-utilisateur :

```
0 0 * * *    root    updatedb
```

Deuxième partie
Introduction à la virtualisation

7 Définition d'un système d'exploitation

7.1 Rôle du système d'exploitation

Un ordinateur possède une architecture matérielle complexe : de la mémoire vive (disposée en une ou plusieurs barrettes), des disques, des périphériques divers et variés (imprimantes, cartes réseaux...). Ces périphériques doivent être accédés par les applications d'une façon relativement "simple" : il n'est pas envisageable de demander aux programmeurs de dialoguer avec un disque dur en langage machine, puis d'organiser lui-même le placement de ses blocs de données sur ce disque. Au contraire, l'application doit accéder à tous les types de disques de la même façon, qu'ils soient branchés en IDE ou en SATA, que le système de fichiers soit ext3 ou ReiserFS (voir paragraphe 4.2). Il en est de même pour tous les périphériques, pour l'utilisation de la mémoire de la machine par les applications, etc.

Sur la plupart des systèmes d'exploitation dits "modernes" (sauf certains systèmes réservés à des usages spécifiques), plusieurs processus sont exécutés par l'ordinateur. On peut voir leur liste en utilisant la commande `ps` et quelques options :

```
coti@maximum:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  10648   720 ?        Ss   avril11   0:18 init [2]
root         2  0.0  0.0      0     0 ?        S    avril11   0:01 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    avril11   0:36 [ksoftirqd/0]
root         6  0.0  0.0      0     0 ?        S    avril11   0:02 [migration/0]
root         7  0.0  0.0      0     0 ?        S    avril11   0:08 [watchdog/0]
root         8  0.0  0.0      0     0 ?        S    avril11   0:00
[migration/1]
[...]
coti      4068  0.0  0.0  14264  5016 tty1     S+   avril11   0:00 -bash
coti      4484  1.8  7.4 2001652 608644 ?     Rl   avril11 1036:01 icedove
[...]
```

On voit que l'on a (beaucoup) plus de processus s'exécutant que l'on dispose de ressources de calcul (c'est-à-dire, ici, de cœurs du processeur). C'est au système d'exploitation que revient la tâche d'arbitrer entre les processus et de leur donner, chacun son tour et pour un temps borné, accès à un cœur. On dit alors qu'il *ordonnance* les processus sur les cœurs de la machine. Il décide d'exécuter un processus pendant un certain temps, puis il le met en attente (état `S` comme *sleeping* dans la sortie de `ps`) en attendant que son tour revienne. Pendant ce temps, il exécute les autres processus un par un de la même façon. C'est par ce mécanisme que l'on a l'impression que plusieurs applications s'exécutent en même temps sur un nombre réduit de cœurs.

Ainsi, on voit que le rôle du système d'exploitation est d'*orchestrer l'accès aux ressources matérielles* des programmes d'exécutant sur la machine. Notamment, il gère l'accès à la mémoire, l'exécution des processus, l'accès aux périphériques via des drivers...

L'architecture en couches d'un tel système est représenté par la figure 12, tirée de l'excellent *Systèmes d'exploitation* d'Andrew Tanenbaum (Pearson Education). Les périphériques physiques sont commandés par le circuit électrique de commandes : la micro-architecture (le chemin composant les registres, l'ALU, etc). Ces circuits sont pilotés par des programmes écrits en langage machine, compréhensible uniquement par cette machine. Ces trois couchent forment le *matériel* de la machine. Ce matériel est exploité par le système d'exploitation, qui nous intéresse ici. Pour faire des appels au système d'exploitation, les programmes, tout en haut de la pile, utilisent des compilateurs ou des interpréteurs de commandes qui transforment un programme compréhensible par un humain en une suite d'instructions compréhensible par la machine.

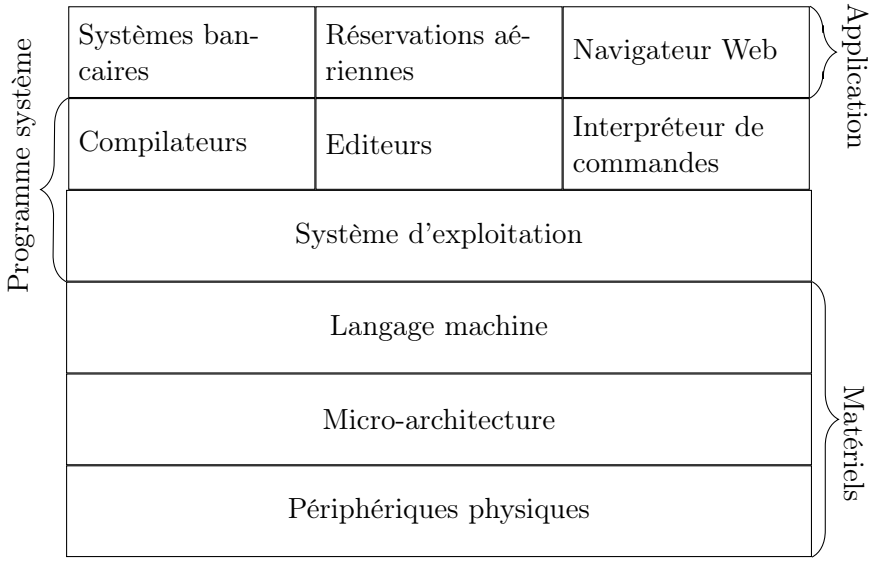


FIGURE 12 – Couches impliquées dans un système informatique : matérielles, système, applications. Schéma issu de Tanenbaum : Systèmes d'Exploitation.

7.2 Le noyau

Au cœur du système d'exploitation, un programme a un rôle particulièrement central : c'est à lui que reviennent les tâches fondamentales du système d'exploitation. On l'appelle le *noyau*.

Au strict minimum, le noyau doit gérer l'accès à la mémoire des processus (à quels endroits de la mémoire un processus a le droit d'accéder) et ordonnancer les processus sur les cœurs du ou des processeur(s). Les autres fonctionnalités reviennent alors à divers programmes et bibliothèques du système d'exploitation.

Vous avez déjà rencontré quelques fonctionnalités du noyau du système d'exploitation. Par exemple, en C on réalise une allocation mémoire avec `malloc()`, et on libère la mémore avec `free ()`. Internement malloc marche en demandant au noyau une certaine quantité de mémoire contiguë pour le processus exécuté. Le noyau examine une sorte de table où il conserve les informations sur l'état de la mémoire et alloue ce qu'il peut. Il retourne (à `malloc`) l'adresse de début de l'espace mémoire qu'il a alloué. `malloc` va utiliser ce bloc de mémoire obtenu depuis le noyau en le divisant en plusieurs tampons—l'idée etant la minimisation des appels au noyau, relativement chers—et renvoyer l'adresse du début de chaque tampon comme résultat pour l'utilisateur.

`free` va recevoir de l'utilisateur l'adresse d'un tampon obtenu alloué par malloc, et il va le « libérer » – au sens de le marquer comme libère. Si *tout* tampon d'un bloc est libéré, `free` va demander au noyau de libérer le bloc, pour réutiliser l'intervalle d'adresses. Après la libération d'un bloc la mémoire correspondante n'est plus accessible par le procès utilisateur, et toute tentative de la lire ou écrire échouera.

Lorsqu'un processus essaye d'accéder à une zone mémoire qui ne lui a pas été allouée, le noyau lui interdit et émet une *erreur de segmentation* : ainsi, ce dernier assure, dans une certaine mesure, un cloisonnement de l'espace mémoire des processus. Ce mécanisme permet de faire en sorte qu'un processus ne puisse pas accéder à l'espace mémoire d'un autre processus (même si des techniques permettent de passer outre).

Pour des raisons de performances, le noyau remplit souvent en pratique beaucoup d'autres fonctionnalités. C'est le cas du noyau Linux. Par exemple, plusieurs piles réseaux (TCP/IP notamment) sont généralement implémentées directement dans le noyau. De même pour un grand nombre de pilotes d'accès au matériel, les systèmes de fichiers...

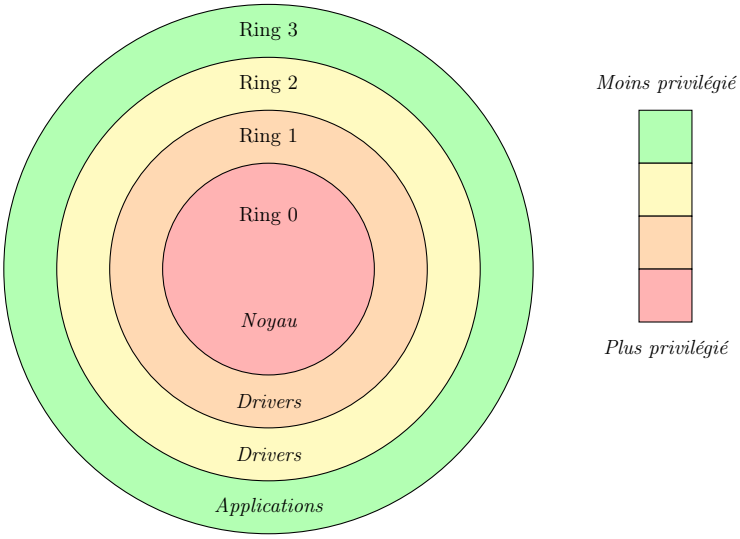


FIGURE 13 – Inclusion des anneaux de privilèges

7.3 Espace noyau vs espace utilisateur

Le noyau a un rôle très critique dans l'exécution des processus sur une machine. Par conséquent, on a besoin de le protéger particulièrement. Pour cela, la plupart des processeurs modernes disposent de *rings d'exécution*. On les traduit souvent en français par "anneaux de privilèges".

Il en existe généralement 4, numérotés de 0 à 3. Plus le numéro de ring est bas, plus il dispose de privilèges d'accès au matériel. Ainsi, le ring 0 a directement accès à toutes les ressources matérielles (et notamment à toute la mémoire), tandis que le ring 3 n'y a pas accès et doit passer par des pilotes ou *drivers* exécutés dans des rings de numéro inférieur.

Les deux rings d'exécution les plus utilisés sont le ring 0, réservé au noyau, et le ring 3, pour les applications. Les rings intermédiaires sont généralement utilisés par les pilotes matériels, qui ont besoin d'avoir accès au matériel et fournissent des fonctionnalités aux applications. La représentation généralement utilisée pour représenter les anneaux d'exécution est présentée sur la figure 13.

Le caractère central du noyau et son rôle de "chef d'orchestre" ont deux conséquences fondamentales :

- Le noyau est le premier programme lancé au démarrage de la machine
- Un seul noyau est exécuté en espace noyau

8 Introduction à la virtualisation

8.1 Pourquoi virtualiser

Aujourd'hui, beaucoup de serveurs sont virtualisés. Il existe plusieurs raisons pour cela. La première est que la puissance des machines est aujourd'hui telle qu'un seul serveur n'exploite pas totalement la machine physique sur laquelle il s'exécute. Ainsi, on exécute plusieurs serveurs sur une seule machine physique.

Ensuite, on pourrait exécuter ces services dans un seul système d'exploitation. Cependant, pour des raisons de sécurité, de stabilité et de facilité d'administration, il est plus simple de n'exécuter qu'un seul serveur par machine. Par exemple, le serveur NFS sera exécuté sur une machine. Le serveur NIS sera sur une autre. Le serveur SVN sera sur une troisième. Le serveur HTTP sera sur un quatrième. Si on décide de regrouper ces quatre serveurs sur une seule machine, il faudra mettre en place un environnement (c'est-à-dire un système d'exploitation, avec notamment des bibliothèques) compatible à la fois avec ces quatre serveurs. Et si un de ces serveurs est corrompu, il existe un risque qu'il corrompe les autres serveurs.

En dédiant un environnement d'exécution, et donc une machine virtuelle, à chaque serveur, chacun est cloisonné et isolé des autres serveurs.

Enfin, la virtualisation apporte des possibilités de migration et de sauvegarde qui améliorent la continuité du service en cas de panne. Par exemple, un serveur exécuté sur une machine physique sur le point de tomber en panne (surchauffe, disque plein...) pourra être migré à chaud et continuer d'assurer son service sans discontinuité ou presque.

8.2 Virtualisation et émulation

L'**émulation** consiste à faire passer une architecture matérielle pour une autre du point de vue de l'application qui s'exécute dessus. Concrètement, cela revient à permettre à une code binaire censé s'exécuter sur une machine A de s'exécuter sur une machine B.

Par exemple, cela permet d'exécuter une application Android sur un ordinateur de développement. On peut citer l'environnement Eclipse qui peut s'interfacer avec un émulateur Android et ainsi tester plus directement le code en développement. Plus ludiquement, on utilise aussi l'émulation pour faire tourner de vieux jeux vidéos prévus pour d'antiques consoles sur des ordinateurs modernes.

L'émulation permet d'exécuter un binaire qui utilise un certain jeu d'instruction et une certaine représentation des données sur un processeur qui utilise un autre jeu d'instructions et/ou une autre représentation des données. Le rôle de l'émulateur est alors de traduire les instructions appelées en instructions disponibles sur le processeur. C'est par conséquent une opération lourde à effectuer.

On peut citer le logiciel QEMU, qui est à la fois un système d'émulation et de virtualisation : il est capable de simuler un système x86, MIPS, ARM, PowerPC ou SPARC.

À l'inverse, la **virtualisation** permet d'exécuter plusieurs machines sur une machine physique. Elle ne fait pas passer un matériel pour un autre. Nous avons vu dans la section 7 que le rôle du noyau du système d'exécution était d'orchestrer l'utilisation des ressources matérielles de la machine et que, par conséquent, une seule instance de ce noyau devait tourner sur une machine mhysique donnée.

La virtualisation permet d'exécuter plusieurs systèmes d'exploitation complets (y compris leur noyau) sur une machine physique donnée. On distingue alors deux catégories de systèmes d'exploitation s'exécutant sur une machine physique :

— Le système hôte, qui a pour rôle cette orchestration des ressources matérielles fournies par la machine. Une seule instance du système hôte s'exécute sur une machine physique.
— Les systèmes invités, qui s'exécutent sur la machine et peuvent être présents en autant d'instances que les capacités de la machine le permettent. Ils s'exécutent en espace utilisateur.

Les ressources de la machine (mémoire, CPU...) sont gérées par le système hôte. Les systèmes invités accèdent à ces ressources ; lorsqu'elles ont besoin d'en obtenir, elles en font la demande au système hôte. Ainsi, la gestion de ces ressources reste centralisée auprès d'un seul noyau : celui du système hôte.

On a l'habitude de classifier les outils de virtualisation en quatre grandes familles en fonction du rôle qu'ils assurent et du confinement qu'ils fournissent entre les systèmes invités :

— Les isolateurs
— Les noyaux en espace utilisateur
— Les machines virtuelles
— Les para-virtualiseurs.

8.3 Système invité et système hôte

Lorsque l'on parle de virtualisation, il est fondamental de faire la distinction entre le système hôte et le système invité. Il faut bien garder à l'esprit le fait qu'**un seul système tourne en espace noyau** : il s'agit du système hôte.

Les systèmes invités, eux, sont exécutés **en espace utilisateur**. Ils sont considérés par l'ordonnanceur comme des processus comme les autres. L'objet des outils de virtualisation et d'émulation est précisément de permettre à ces systèmes invités d'être exécutés en espace utilisateur.

8.4 Confinement sur disque avec `chroot`

L'utilitaire `chroot` est un isolateur. Il permet de ne donner à une application qu'une vision limitée de l'arborescence du système de fichiers. Ainsi, sur le disque dur, l'application est confinée des autres applications.

Il permet de ne pas donner accès à l'intégralité du disque dur à une application. Ainsi, on peut exécuter sur un système plusieurs instances de certaines applications censées s'exécuter seules sur le système. Il n'assure une isolation qu'au niveau du disque dur : on ne peut par exemple pas exécuter plusieurs noyaux en utilisant `chroot` seul.

Il est souvent utilisé comme première précaution de sécurité pour des serveurs comme des serveurs Web ou FTP. Le but est de ne pas donner accès à tout le disque à une application qui apporte un point d'accès à la machine depuis l'extérieur. Attention cependant, il n'offre qu'un niveau de sécurité assez bas. Il est relativement facile de sortir d'un environnement chrooté.

Le principe de `chroot` est de déplacer la racine du système de fichier à un endroit arbitraire de son arborescence. Or, nous savons qu'il est impossible de remonter plus haut dans l'arborescence qu'au niveau de sa racine. Ainsi, si l'on se place à la racine de l'arborescence et que l'on demande à remonter dans le répertoire parent du répertoire courant, on reste à la racine :

```
coti@maximum:~$ cd /
coti@maximum:/$ pwd
/
coti@maximum:/$ cd ..
coti@maximum:/$ pwd
/
```

Par exemple, exécutons un interpréteur de commandes (un shell, par exemple `/bin/bash`) dans un environnement dont nous déplaçons la racine dans le répertoire `/var/www`. Vous verrez en TP comment préparer un environnement pour `chroot`.

```
coti@maximum:~$ pwd
/users/coti
coti@maximum:~$ sudo chroot /var/www /bin/bash
bash-4.2# pwd
/
bash-4.2# cd ..
bash-4.2# pwd
/
bash-4.2#
```

L'application exécutée (`/bin/bash`) ne peut pas remonter plus haut que la racine de la sous-arborescence de fichiers à laquelle elle a accès, qui est en réalité `/var/www`.

On peut représenter cette vision partielle avec les figures 14 et 15.

8.5 Exécution d'un noyau en espace utilisateur

Avec l'utilitaire du noyau *UML* (User-Mode Linux) on peut, comme son nom l'indique, exécuter un noyau Linux en espace utilisateur. Son but était à l'origine de permettre le développement, le test et le débuggage du noyau. Ainsi, il est possible de lancer autant de noyaux Linux sur une machine et de tester leurs fonctionnalités. Le logiciel Marionnet utilise des machines virtuelles UML.

Le noyau invité est compilé spécifiquement pour être un exécutable comme un autre, et il est lancé sur la machine comme n'importe quel programme. Il doit également recevoir en option un système de fichiers, qui peut être un fichier sur lequel est installé un système de fichiers.

Le noyau se lance comme un exécutable. On peut lui passer en paramètres des systèmes de fichiers (et même une partition swap) en précisant sur quel device ce système de fichier sera accessible depuis la machine virtuelle. Il faut également lui spécifier quel device sera la racine du système de fichiers, et la quantité de mémoire vive sera attribuée à la machine virtuelle.

```
root@maximum:/tempo# ./linux-3.11.0-uml ubda=disk.img ubdb=swap.img \
       root=/dev/ubda mem=256M
```

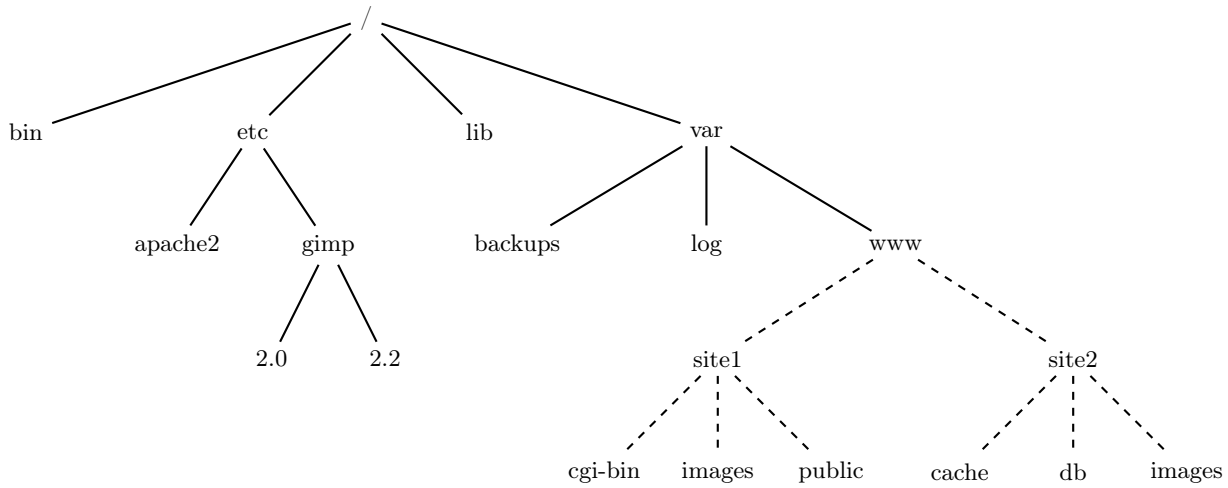


FIGURE 14 – Arborescence de fichiers : le `chroot` va être fait dans le répertoire `/var/www`.

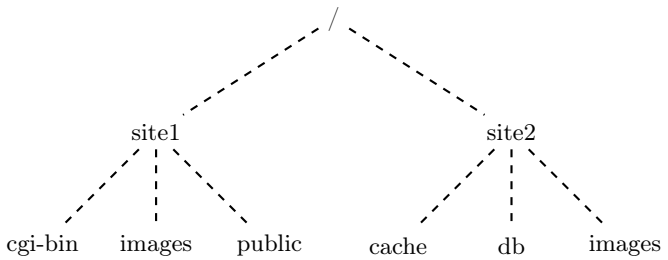
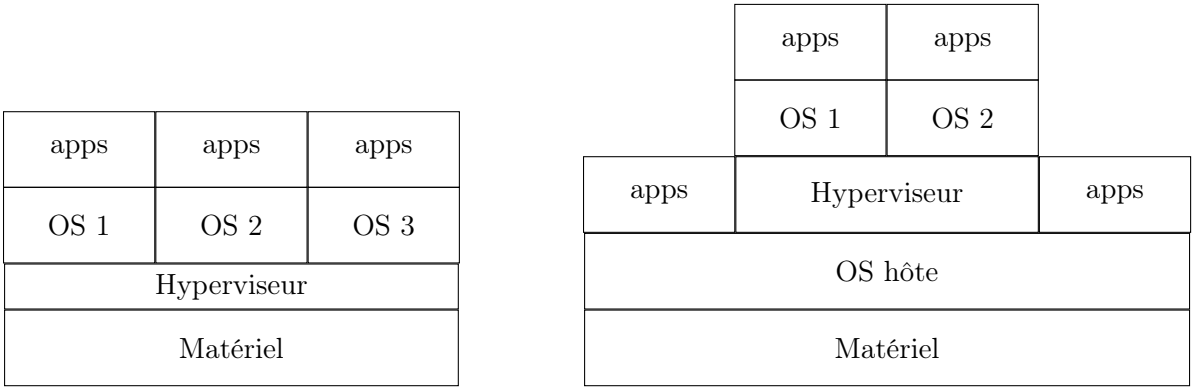


FIGURE 15 – Arborescence de fichiers vue dans l'environnement du `chroot` : seul un sous-arbre est vu, la racine étant déplacée.



(a) Hyperviseur de type I (b) Hyperviseur de type II

FIGURE 16 – Position des deux types d'hyperviseurs et des noyaux hôtes et invités.

La machine virtuelle ainsi exécutée aura pour noyau le noyau invité (exécuté en espace utilisateur par UML) et comme partition principale le système de fichiers passé en paramètre au lancement.

L'utilitaire `pstree` permet de voir les processus sous une forme arborescente : chaque processus est rattaché à son processus parent, le premier processus étant le processus `init`, lancé au démarrage par le noyau. Si on lance une machine virtuelle UML dans Marionnet, on voit, on obtient, parmi la sortie de `pstree` :

```
--gnome-terminal---bash---marionnet-------linux-2.6.18-gh--|----15*[linux-2.6.18-gh]
                                          |                |----xterm-----port-helper
                                          |--2*[vde_switch]
                                          |--wirefilter
                                          |-10*[{marionnet}]
```

Depuis le processus `gnome-terminal`, on a lancé l'interpréteur de commandes (`bash`, qui lui-même a lancé `marionnet`. Depuis `marionnet` a été lancé un noyau UML d'où on voit, entre autres, `xterm` et `port-helper`.

UML permet aussi d'accéder au réseau ou de créer un réseau virtuel sur la machine, permettant ainsi de relier sur ce réseau virtuel plusieurs machines virtuelles UML.

Par conséquent, il permet une relativement bonne isolation de ce qui est exécuté dans la machine virtuelle. Cependant, il offre des performances relativement médiocres, notamment en n'assurant pas une bonne isolation de l'usage des ressources de la machine physique par les différentes machines virtuelles.

8.6 Paravirtualisation

Un **hyperviseur** est un noyau hôte allégé dont le seul rôle est d'exécuter des noyaux invités. Il existe deux types d'hyperviseurs :

— Hyperviseur de *type I*, ou natif, ou bare metal : il est exécuté directement sur le matériel de la machine. Il s'agit d'une fine couche logicielle située entre le matériel et le système. Tous les noyaux s'exécutant dessus sont des noyaux invités.

— Hyperviseur de *type II*, ou hébergé : il se situe entre le système d'exploitation hôte et les systèmes invités.

On peut voir une représentation de la position de ces deux types d'hyperviseurs sur la figure 16.

Les noyaux invités sont tous exécutés *en espace utilisateur* : soit dans les rings 1 et 2, soit dans le ring 3. Ils ne peuvent donc pas avoir accès à toutes les ressources de la machine. Lorsqu'ils en ont besoin, ils effectuent un appel à l'hyperviseur qui, lui, tourne dans le ring 0. Par conséquent, le noyau du système invité doit être modifié : on ne peut pas faire tourner des systèmes d'exploitations tels qu'ils sont d'origine lorsqu'ils tournent en tant que systèmes d'exploitation hôtes ou seuls.

Sous Linux, on peut citer Xen ou KVM, qui est un hyperviseur de type 1. Ce dernier est intégré au noyau Linux : il en fait un "gros hyperviseur". Le noyau hôte est l'hyperviseur, et il fait tourner des noyaux invités.

À l'inverse, VirtualBox, QEMU et VMware Workstation sont des hyperviseurs de type II : ils tournent à l'intérieur du système d'exploitation hôte.

Les *Linux Containers* (lxc) sont un système de virtualisation légère orienté vers le cloisonnement des systèmes invités. Ils reposent sur un mécanisme de contrôle appelé les *cgroups*. Grâce à ce système, chaque système invité (appelé un conteneur) a accès uniquement aux ressources qui lui sont attribuées. Par exemple, il a son propre système de fichiers, son propre espace mémoire inviolable par les autres conteneurs... Le cloisonnement est fait au niveau du noyau hôte, donc en espace noyau.

9 Introduction au cloud computing

On parle aujourd'hui beaucoup de *Cloud Computing*, ou informatique dans le nuage. C'est souvent présenté comme la solution à tous les problèmes des entreprises. Le but de cette section est de présenter quelques aspects concrets du cloud.

9.1 Principes du cloud

Le principe de base du cloud est l'*externalisation de ressources*. Un fournisseur de service apporte un service. Il peut s'agir d'un service de stockage, comme dans le cas d'un cloud de stockage, ou encore d'un cloud applicatif.

Les ressources matérielles sont situées chez le fournisseur de services, qui les localises où il le souhaite. Le client n'a lui pas besoin de savoir où sont situées ces ressources. Ces ressources peuvent même être disséminées à plusieurs endroits : c'est ce qu'on appelle un cloud distribué, comme pour celui de Vifib.

Un des premiers clouds commerciaux a été EC2 : le Elastic Compute Cloud d'Amazon. L'idée de base est de fournir, à la demande, des machines de calcul à des clients. Sur chaque machine tourne un hyperviseur Xen. Les utilisateurs peuvent déployer une machine virtuelles sur les ressources attribuées par EC2, et exécutent l'environnement de leur choix sur celles-ci. Ainsi, un utilisateur qui voudra faire tourner un code parallèle de calcul scientifique nécessitant un certain nombre de bibliothèques spécifiques pourra préparer son environnement sur une machine virtuelle qui sera déployée sur les machines qu'il louera dans le cloud EC2.

Le cloud est actuellement très apprécié du fait de sa flexibilité : il nécessite pour l'entreprise qui y fait appel relativement peu d'investissement en matériel (pas d'achat de matériel) et en personnel (administration système réduite). De plus, il a une certaine élasticité qui permet de s'adapter rapidement à un changement des besoins. Si du jour au lendemain les besoins de l'entreprise en stockage sont multipliés par 10, tant que le fournisseur de service le permet, il suffit de modifier les termes du contrat de service en "louant" davantage d'espace.

9.2 Modèle économique

Le modèle économique du cloud se base sur la notion de service. Le client (l'entité qui fait appel au service) émet une demande : il peut vouloir un certain nombre de cœurs, un certain espace de stockage, un certain nombre d'applications.

Il conclut alors un accord de service, ou *Service Level Agreement*, dont les termes définissent le niveau de service fourni. Notamment, la qualité et la résilience du service. Par exemple, le client peut demander à avoir un certain nombre de machines pendant la durée d'accès au cloud. Si certaines de ces machines tombent en panne, charge au prestataire de service de les remplacer, voire d'assurer la continuité du service. On définit alors le niveau de résilience. Si plus de machines tombent en panne que le nombre spécifié dans le SLA, ou que le service est interrompu pendant plus de temps, le contrat n'est pas respecté et des pénalités s'appliquent au prestataire de service.

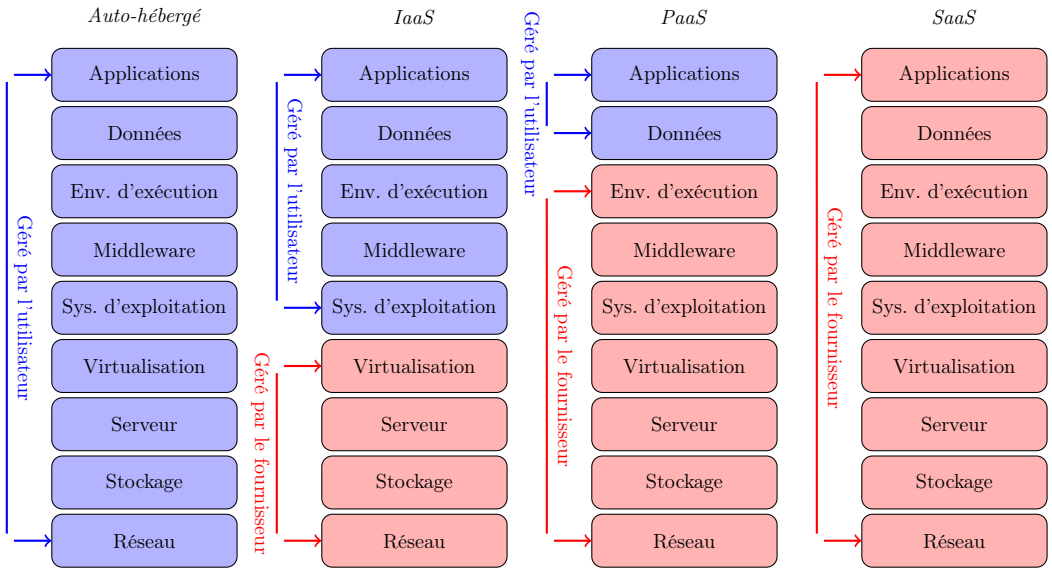


FIGURE 17 – Piles logicielles correspondant aux différents services de cloud

9.3 Types de cloud

La figure 17 présente les différents types de cloud existant et les piles logicielles qui leur sont associées ainsi que la responsabilité de la gestion des différentes couches.

Tout à gauche, on a le cas particulier du service **auto-hébergé** : l'utilisateur dispose des machines dans ses locaux, il les administre et il a le contrôle de l'environnement d'exécution et des applications. Il est le seul à intervenir sur ces machines.

Dans le cas de l'**Infrastructure as a Service** (IaaS), comme son nom l'indique, c'est dans l'infrastructure matérielle que réside le service. Le prestataire propose des machines physiques e une couche de virtualisation. Le client apporte ses machines virtuelles et il contrôle toutes les couches logicielles à partir du système d'exploitation invité. C'est par exemple le cas du cloud d'Amazon EC2.

Le cloud de type **Platform as a Service** fournit un environnement d'exécution. L'utilisateur (le client du service) est en charge, lui, de s'occuper des données et de l'application. C'est par exemple le cas des services d'hébergement Web. Le client met sur le serveur son site Web (l'application) et ses données. Le prestataire fournit un environnement d'exécution, c'est-à-dire un serveur Web qui va exécuter ce site et stocker les données.

Enfin, le service le plus abstrait est le **Service as a Service** (SaaS). Ici, ce qui est vendu est un service : par exemple, un blog, une application... On put citer beaucoup de services de Google dans ce cas : les Google docs, l'interface Gmail... On exécute de réelles applications directement dans le navigateur.

9.4 Évolution : vers du tout-décentralisé

On observe aujourd'hui une grande externalisation des données et des applications. Beaucoup d'entreprises externalisent le stockage de leur données pour faire appel à une prestation de stockage mais aussi de sauvegarde et d'accès depuis n'importe où. ces services sont aujourd'hui accessibles également par les particuliers. Par exemple, des données stockées sur un service de stockage comme le Google Drive ou Dropbox sont accessibles depuis n'importe où. Dans un contexte d'utilisation mobile, où les utilisateurs changent de terminal (d'un ordinateur personnel à la maison à un ordinateur professionnel, par exemple), ces données restent accessibles de façon transparente.

On observe en même temps un allègement des terminaux utilisateurs. C'est le cas notamment des tablettes et des netbooks : afin de gagner en poids, autonomie et coût, on fabrique des machines ayant des capacités et des performances réduites. Le stockage local est relativement réduit, et les données

sont principalement stockées sur un service de cloud.

De même, on se dirige vers des applications accessibles directement dans le navigateur. Par exemple, un utilisateur des Google Docs trouvera, depuis n'importe quel ordinateur relié à Internet, son environnement de travail habituel dans son navigateur.

On revient à une tendance des années 1950 à 1970 où, par la force des choses, on ne pouvait pas se permettre de déployer autant d'ordinateurs puissants que de postes de travail. On avait alors un très gros ordinateur central, puissant, appelé *mainframe*. Les postes de travail étaient, eux, des *clients légers*. Les clients étaient reliés au mainframe par le réseau local. Les données étaient stockées sur le mainframe, qui exécutait aussi les applications. Les clients légers, eux, n'avaient en charge que l'affichage.

Le cloud retourne vers cette tendance, cette fois-ci à l'échelle d'Internet. Les terminaux de plus en plus mobiles (smartphones, tablettes) n'ont plus les ressources permettant d'assurer toutes les fonctionnalités demandées, qui sont externalisées dans un service distant, qu'on nomme de façon abstraite "le cloud".